

Project 2 Discussion

CS 211



**COMPUTER
SCIENCE
COLLEGE OF
ENGINEERING**

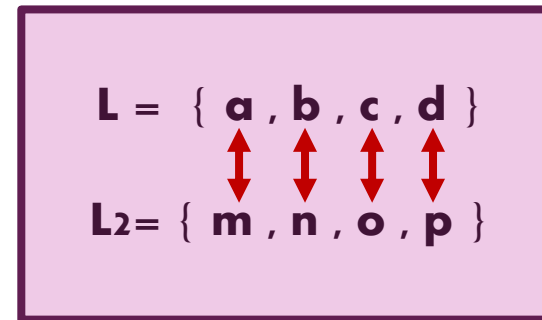


Decoding an Encrypted Message

- Alice and Bob need to communicate with encryption
- They have two sets of alphabet, one is the main alphabet that conveys messages and the other is a dummy alphabet

- $L = \{a, b, c, d\}$
- $L_2 = \{m, n, o, p\}$

- L_2 maps to L



Decoding an Encrypted Message


- Each message they send is an array of strings
- Only some words are meaningful
- They have an agreement for decoding messages
- A string is meaningful if every existing letter from L is mapped/covered by corresponding letter from L2 in proper order.
- The mapping **MUST** properly follow the sequence of letters.
- A meaningful string **MUST** have a letter from L.



Decoding an Encrypted Message

Proper Nesting of letters is required

abcghonm ? → abcghonm Valid ✓



The diagram shows the string 'abcghonm' with a red question mark. An arrow points to the same string where each letter is a different color: a (blue), b (purple), c (yellow), g (red), h (cyan), o (green), n (orange), m (dark blue). Brackets indicate proper nesting: a yellow bracket above 'cgh' and a red bracket below 'ghon'.

dcayuum ? → dcayuum Invalid ✗




The diagram shows the string 'dcayuum' with a red question mark. An arrow points to the same string where each letter is a different color: d (blue), c (purple), a (yellow), y (red), u (cyan), o (green), m (orange). A red bracket below 'ayuo' indicates invalid nesting.

Decoding an Encrypted Message

Number of letters from L2 = Number of letters from L

abcghonmpm ? \rightarrow abcghonm Valid ✓



abcghon ? \rightarrow abcghon Invalid ✗






bacghnmop ? \rightarrow bacghnmop Invalid ✗



Decoding an Encrypted Message

Letters from L should occur before letters from L2

abcghonm ?   Valid 

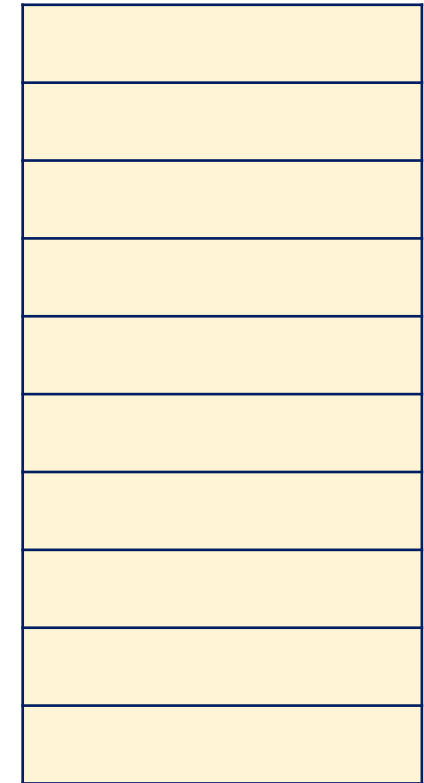
mnotycba ?  mnotycba Invalid 

Stack Data Structure

- Stack is a linear data structure that follows the LIFO (Last In First Out) rule
- Think of stacking plates
- The top of stack is always the most recent element that was pushed into the stack
- Stack supports these operations:
 - push() - Insert a value on top
 - pop() - Remove the value from the top
 - top() - return the value of the top element

Stack Data Structure

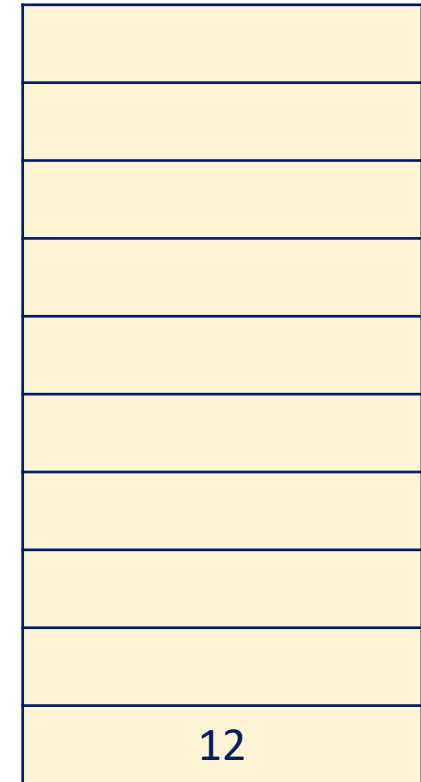
- Operations on stack



Stack

Stack Data Structure

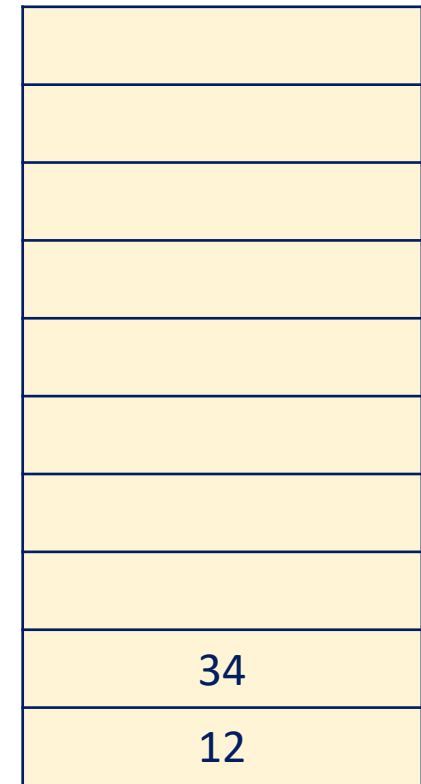
- Operations on stack
 - push(12)



Stack

Stack Data Structure

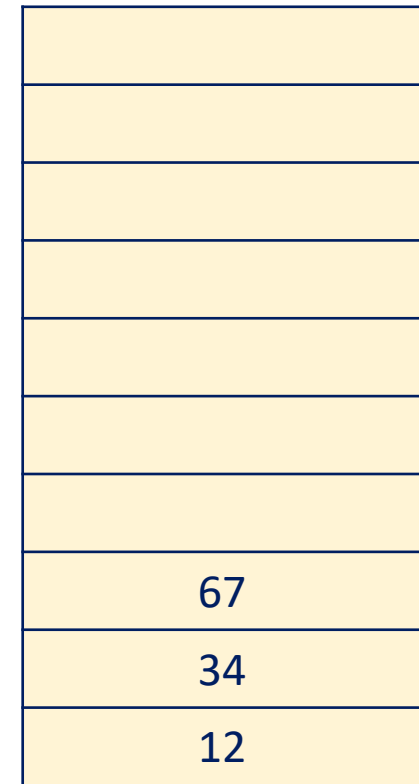
- Operations on stack
 - push(12)
 - push (34)



Stack

Stack Data Structure

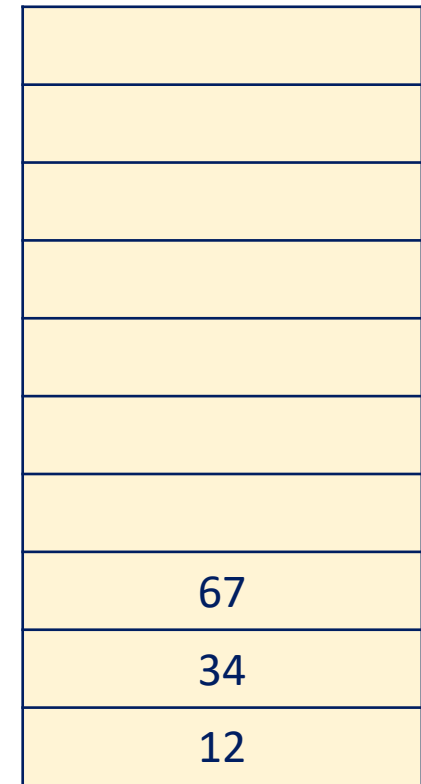
- Operations on stack
 - push(12)
 - push (34)
 - push(67)



Stack

Stack Data Structure

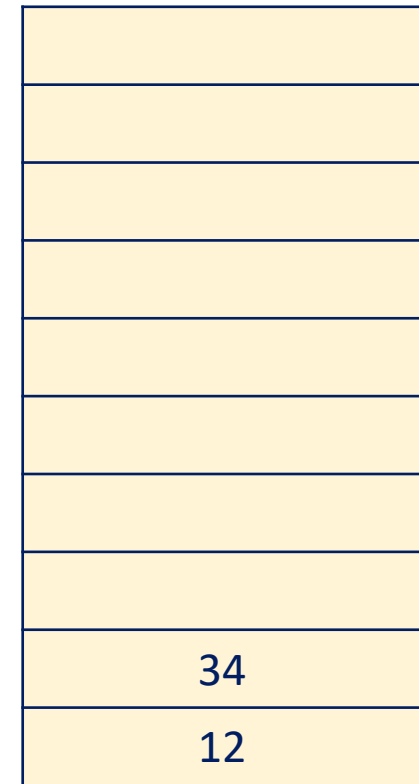
- Operations on stack
 - push(12)
 - push (34)
 - push(67)
 - top() → 67



Stack

Stack Data Structure

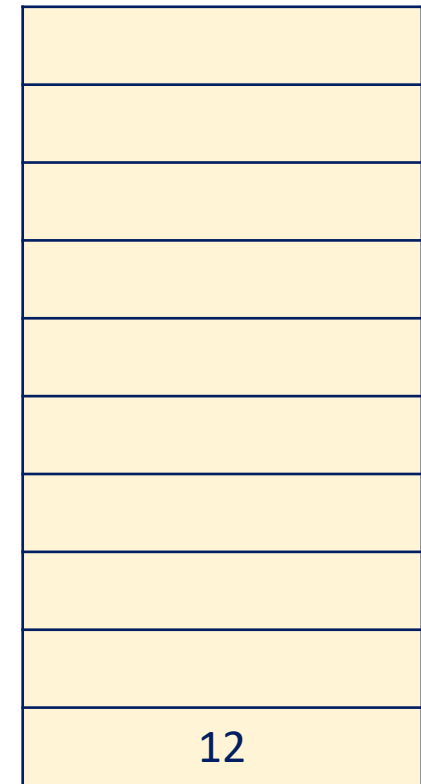
- Operations on stack
 - push(12)
 - push (34)
 - push(67)
 - top() → 67
 - pop()



Stack

Stack Data Structure

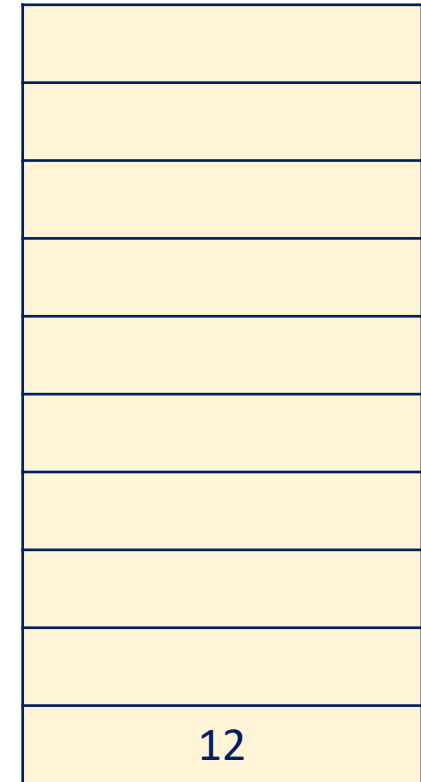
- Operations on stack
 - push(12)
 - push (34)
 - push(67)
 - top() → 67
 - pop()
 - pop()



Stack

Stack Data Structure

- Operations on stack
 - push(12)
 - push (34)
 - push(67)
 - top() → 67
 - pop()
 - pop()
 - top() → 12

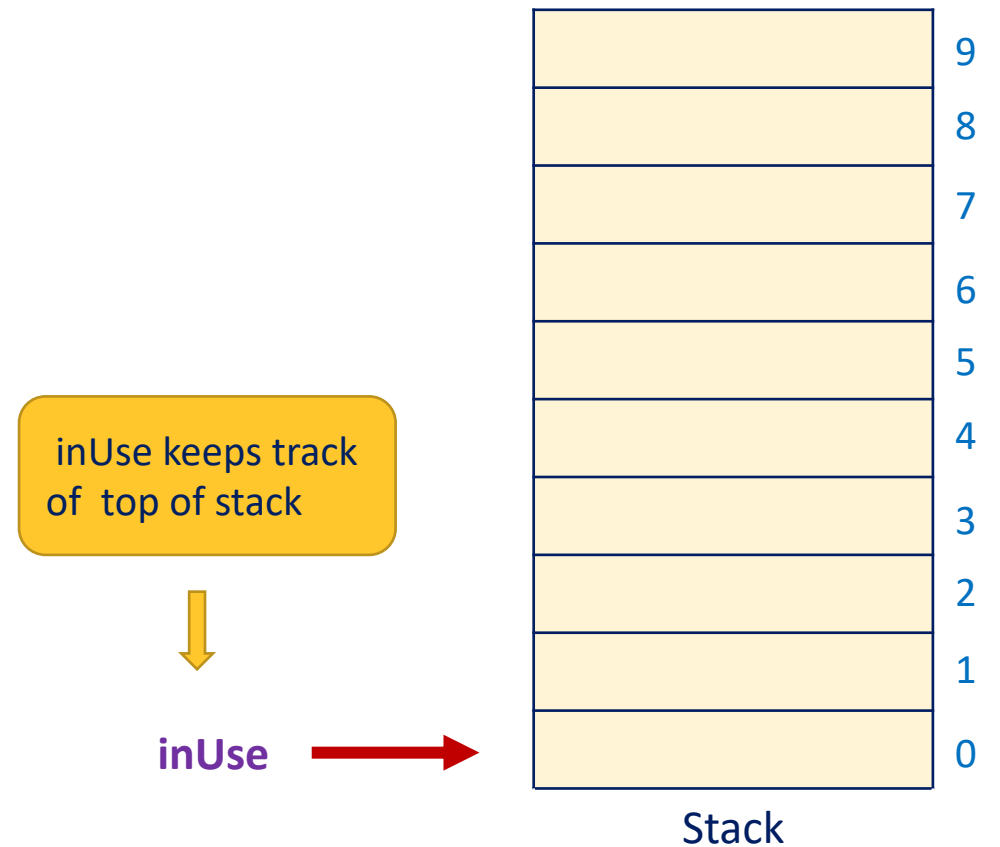


Stack

Stack Data Structure

- Operations on stack

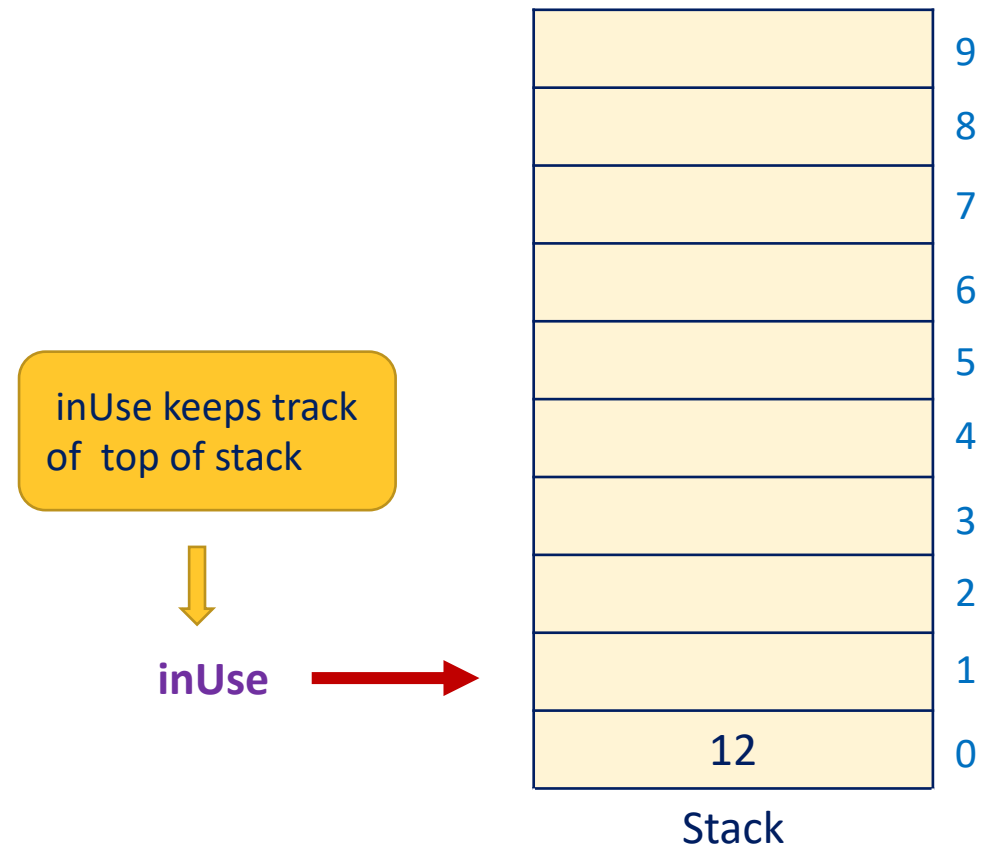
We can implement a stack using dynamic array struct



Stack Data Structure

- Operations on stack
 - push (12)

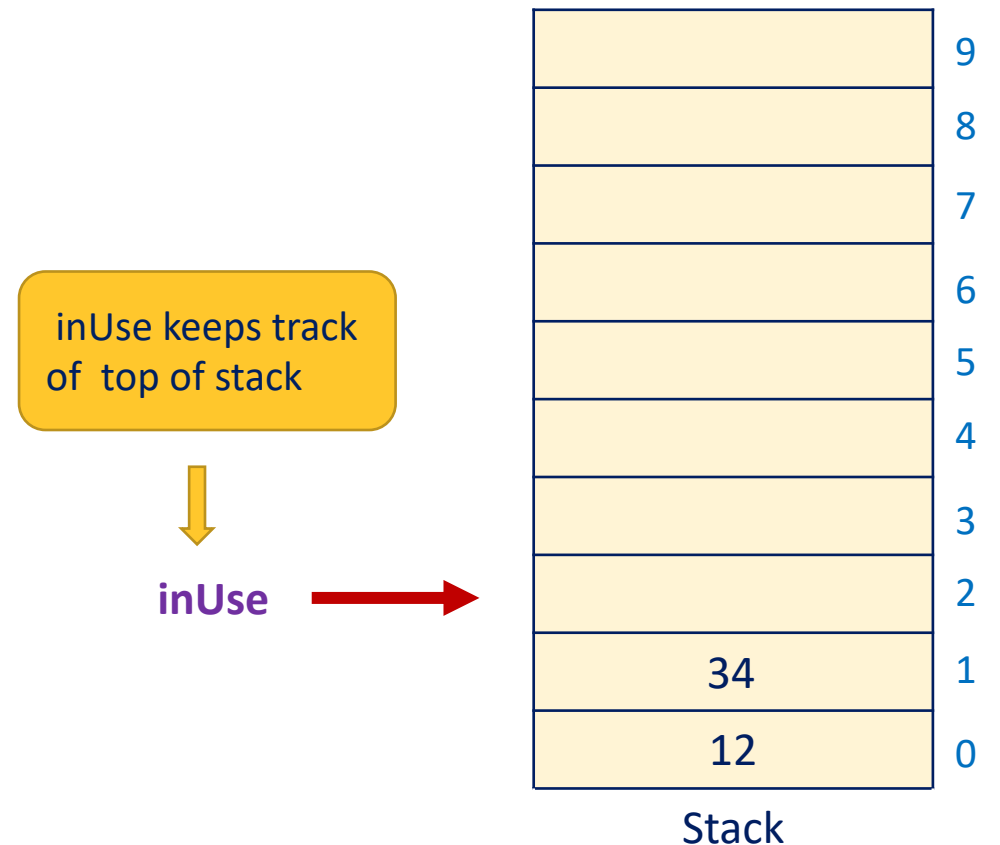
We can implement a stack using dynamic array struct



Stack Data Structure

- Operations on stack
 - push (12)
 - push (34)

We can implement a stack using dynamic array struct

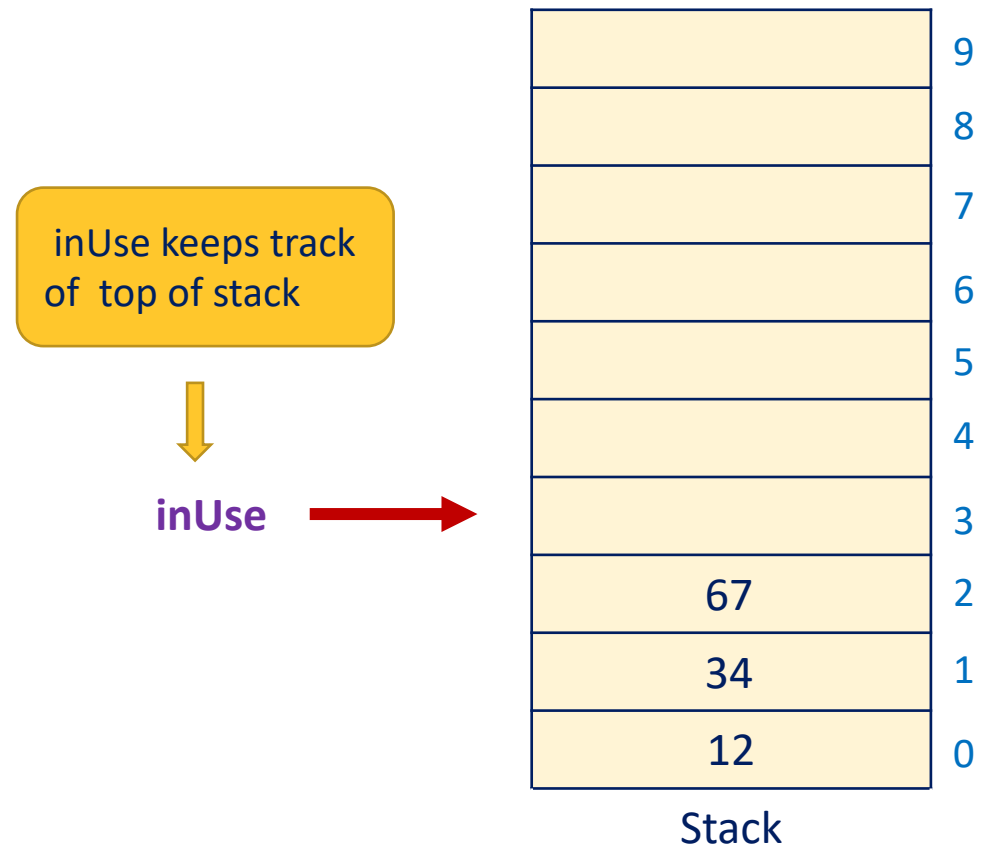


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)

We can implement a stack using dynamic array struct

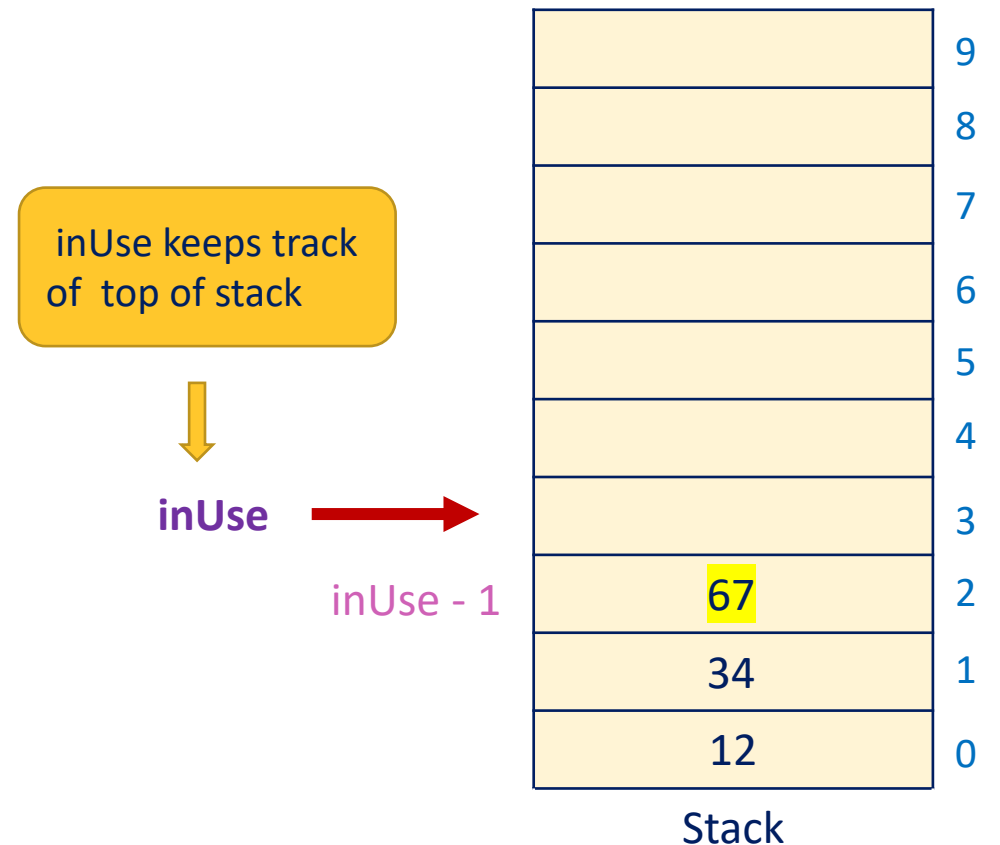


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67

We can implement a stack using dynamic array struct

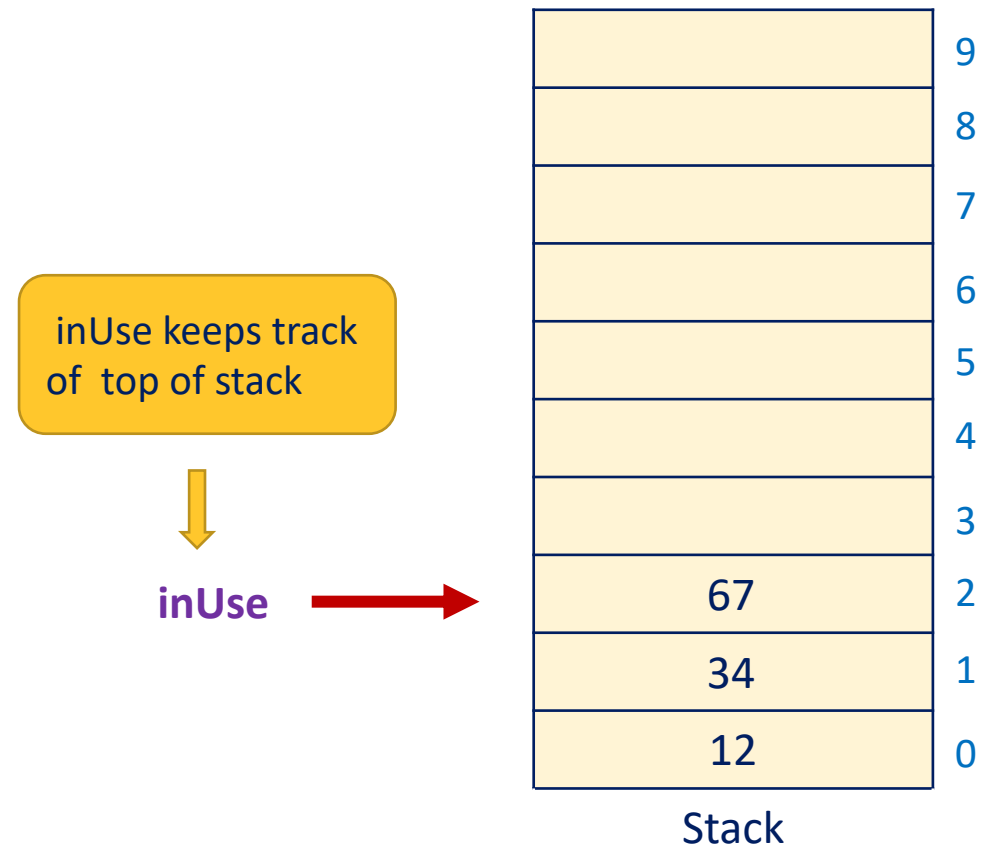


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67
- pop ()

We can implement a stack using dynamic array struct

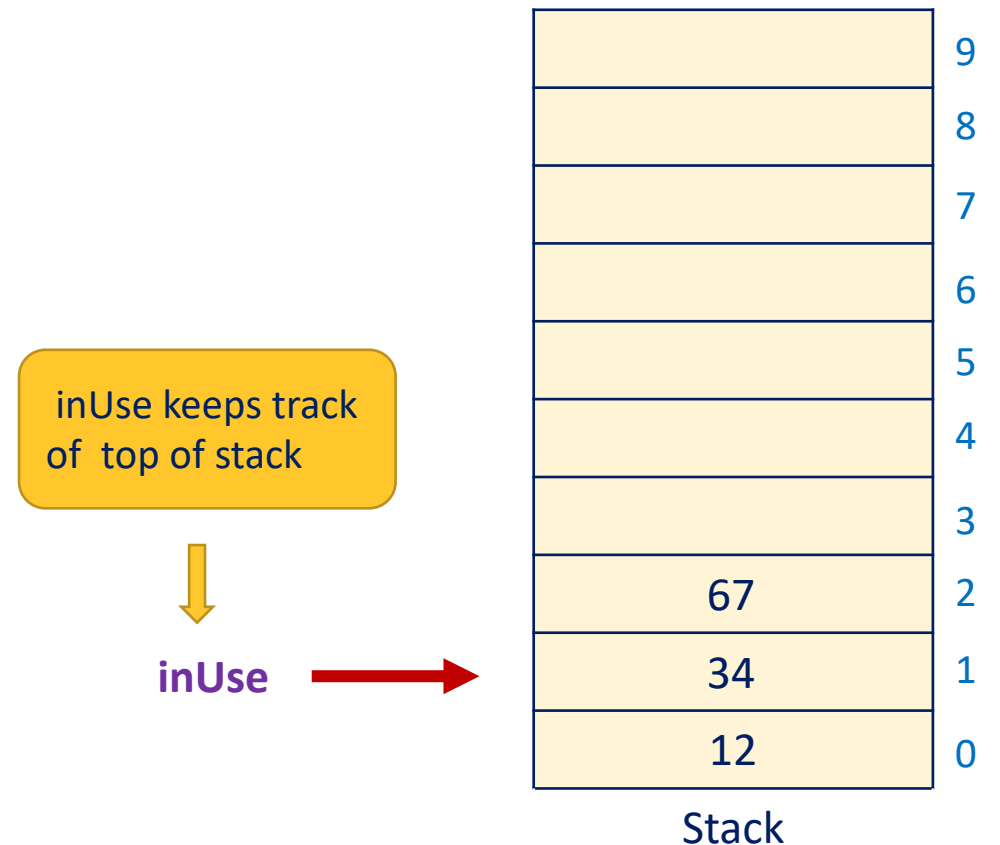


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67
- pop ()
- pop ()

We can implement a stack using dynamic array struct

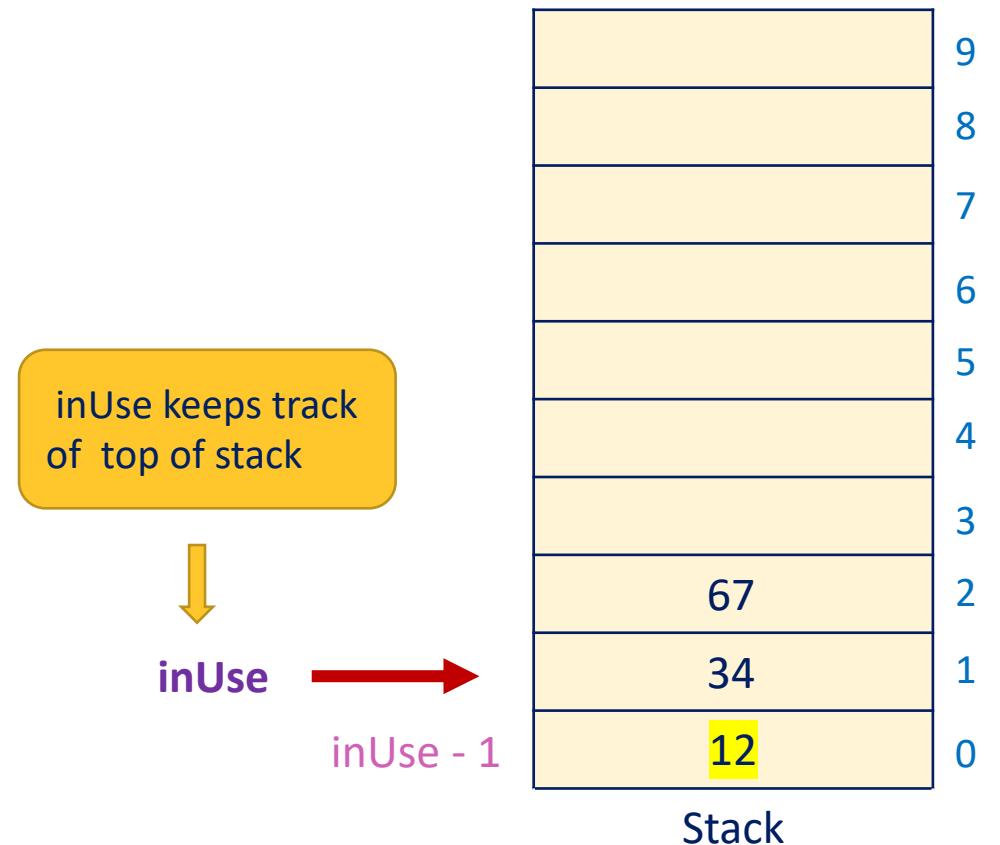


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67
- pop ()
- pop ()
- top () => **inUse-1** => 12

We can implement a stack using dynamic array struct

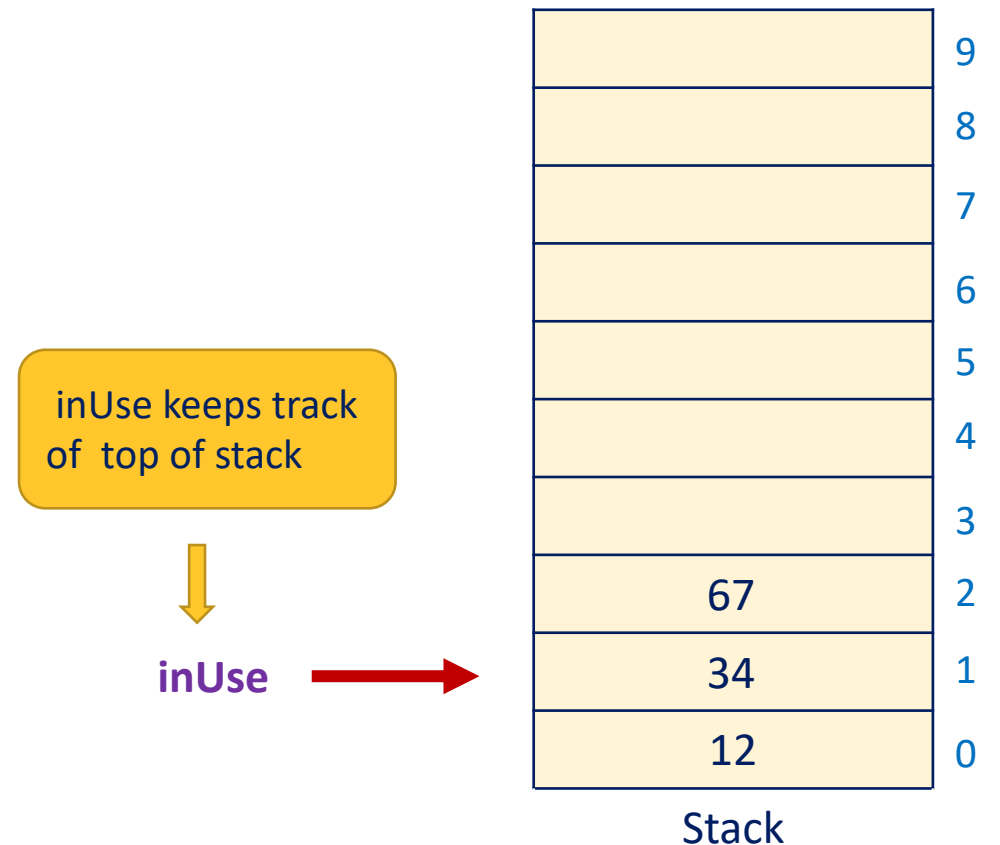


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67
- pop ()
- pop ()
- top () => **inUse-1** => 12
- push(20)

We can implement a stack using dynamic array struct

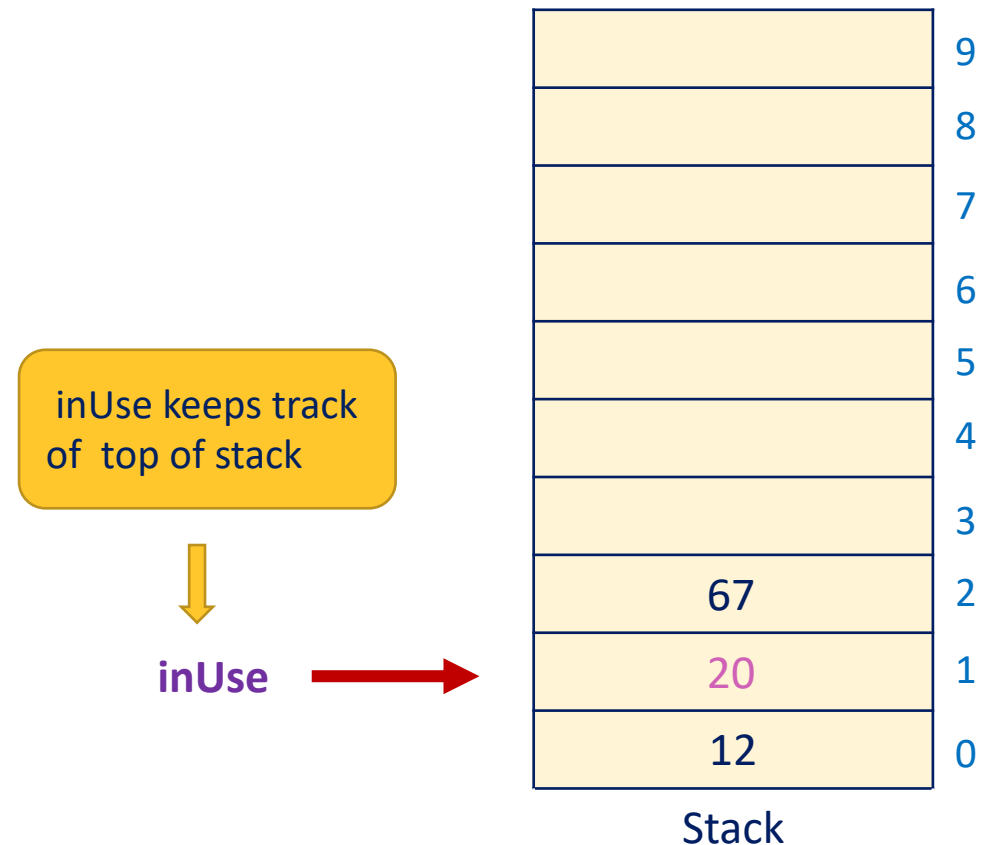


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67
- pop ()
- pop ()
- top () => **inUse-1** => 12
- push(20)

We can implement a stack using dynamic array struct

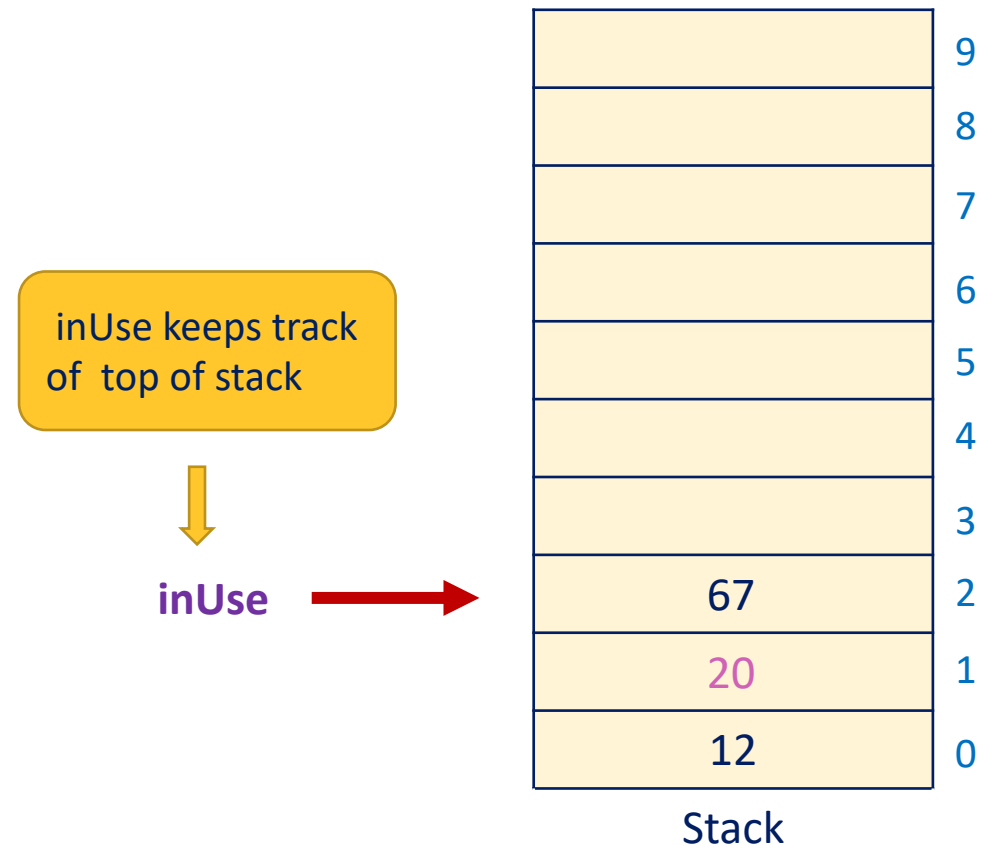


Stack Data Structure

- Operations on stack

- push (12)
- push (34)
- push (67)
- top () => **inUse-1** => 67
- pop ()
- pop ()
- top () => **inUse-1** => 12
- push(20)

We can implement a stack using dynamic array struct



General Algorithm

If your current character is a member of L:

- push the character into the stack

If your current character is a member of L2:

- Verify the top of the stack contains the matching letter from L2
- pop/remove the letter from L from the stack

When you encounter the end of the string

- Verify the stack is empty => String is meaningful and part of the message

Ignore any other symbols contained in the expression



General Algorithm

You check for invalid string when:

Your current character is a member of L2:

- The stack is empty \Rightarrow you have extra letters from L2
- The top of stack letter is not a correct match

When you encounter the end of the string

- If the stack is not empty \Rightarrow you have extra letters from L that are not mapped

If a string is invalid, you can move to the next string in the message



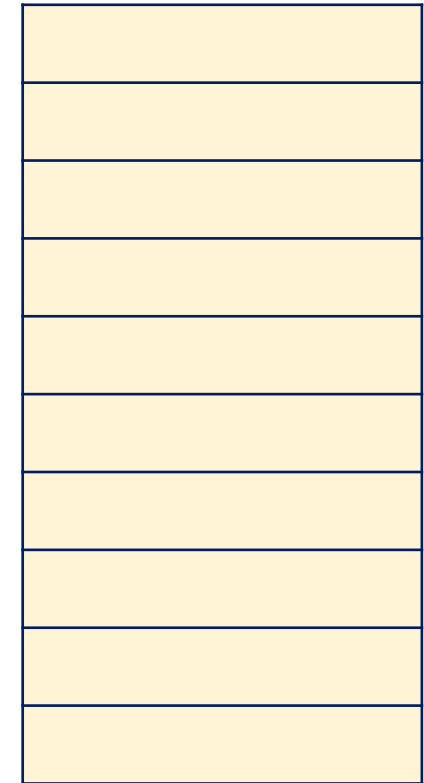
Decoding a Valid String

String: a b c g o n m

Current Character: At start of Expression

Character Type: Nothing yet

Task: Clear stack



Stack

Decoding a Valid String

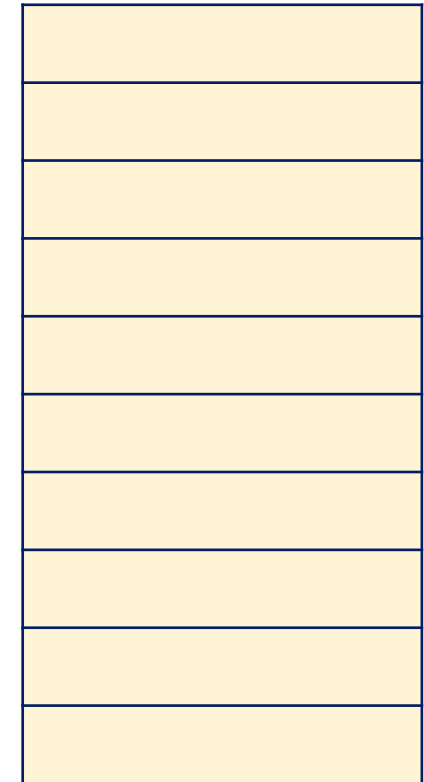
String: a b c g o n m

^

Current character: a

Character Type: Belongs to **L**

Task: push onto stack



Stack

Decoding a Valid String

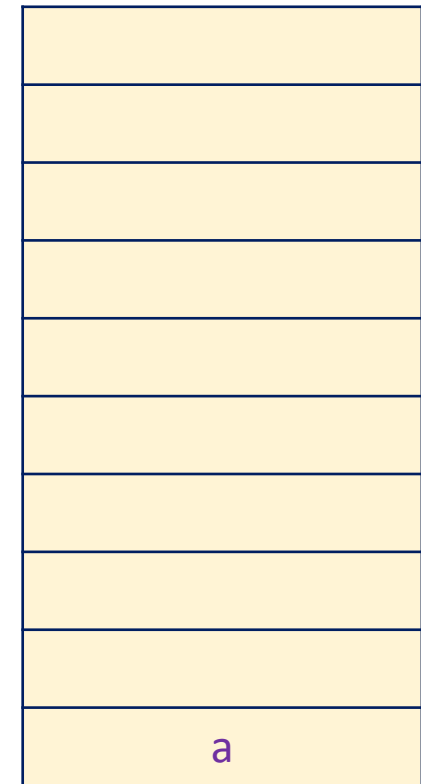
String: a b c g o n m

^

Current character: a

Character Type: Belongs to **L**

Task: push onto stack



Stack

Decoding a Valid String

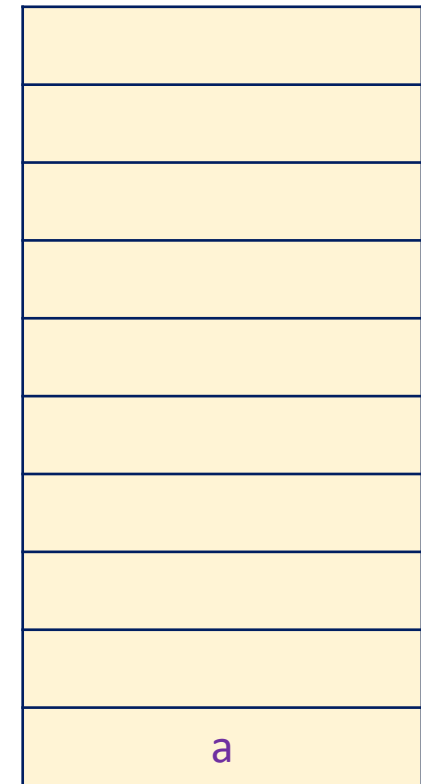
String: a b c g o n m

^

Current character: b

Character Type: Belongs to **L**

Task: push onto stack



Stack

Decoding a Valid String

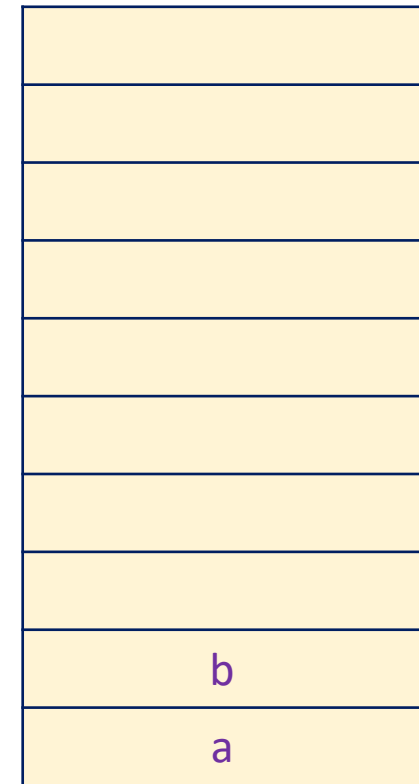
String: a b c g o n m

^

Current character: b

Character Type: Belongs to **L**

Task: push onto stack



Stack

Decoding a Valid String

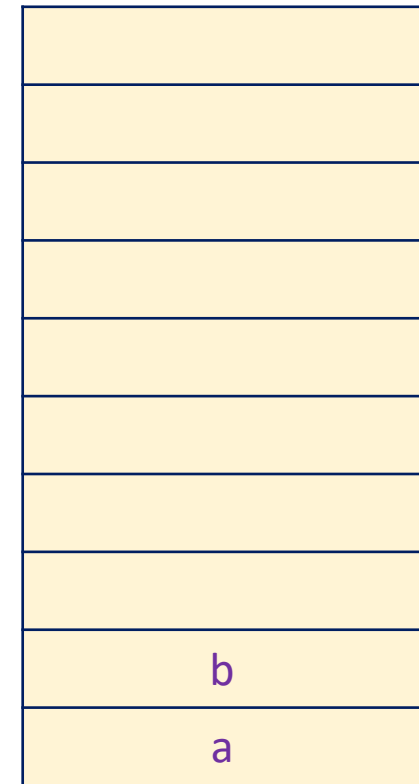
String: a b c g o n m

^

Current character: c

Character Type: Belongs to **L**

Task: push onto stack



Stack

Decoding a Valid String

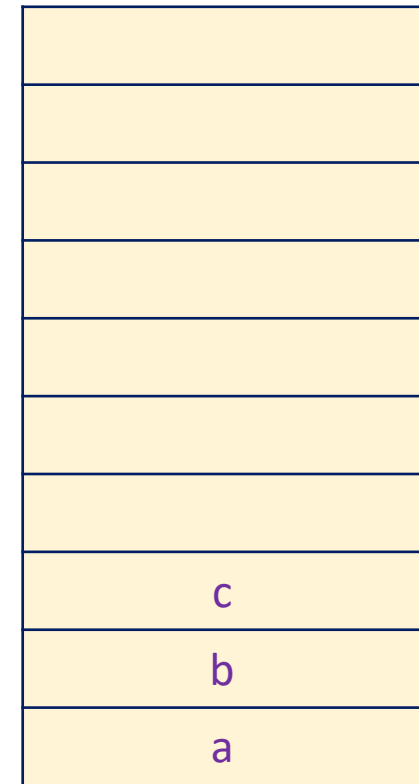
String: a b c g o n m

^

Current character: c

Character Type: Belongs to **L**

Task: push onto stack



Stack

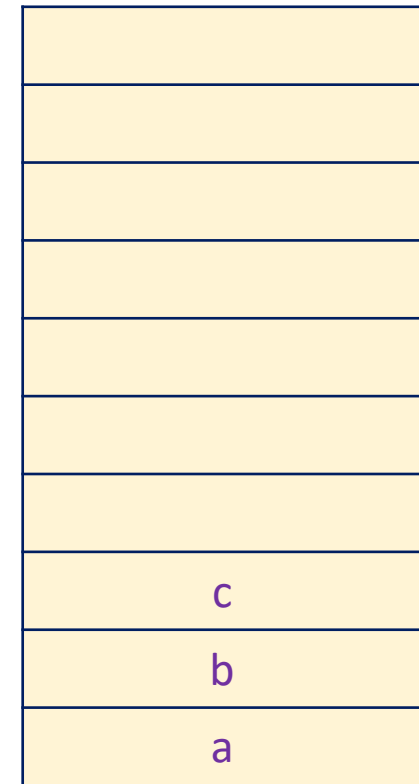
Decoding a Valid String

String: a b c g o n m
 ^

Current character: g

Character Type: Not in ***L*** or ***L2***

Task: ignore



Stack

Decoding a Valid String

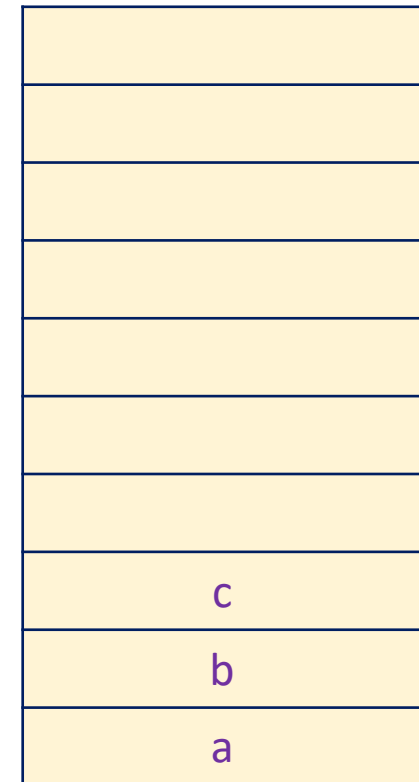
String: a b c g o n m

^

Current character: o

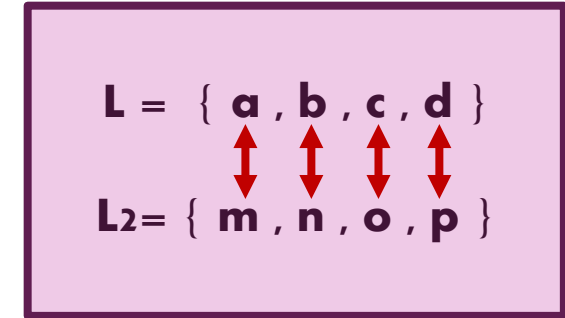
Character Type: Belongs to **L2**

Task: Verify if it is mapping top of stack
 character, if so pop the stack



Stack

Decoding a Valid String



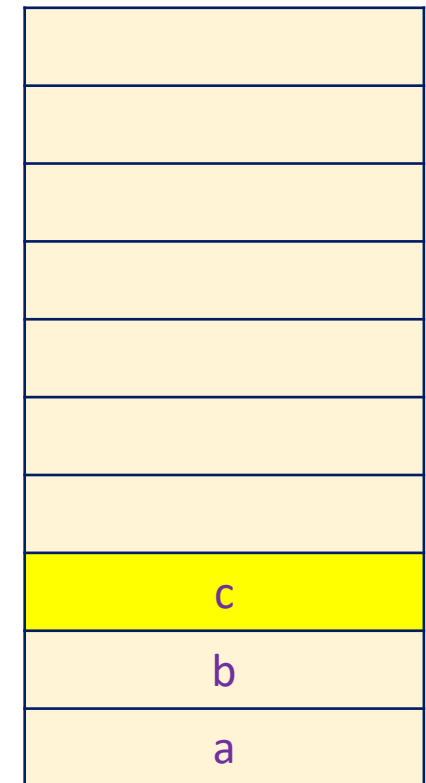
String: a b c g o n m

^

Current character: o

Character Type: Belongs to ***L2***

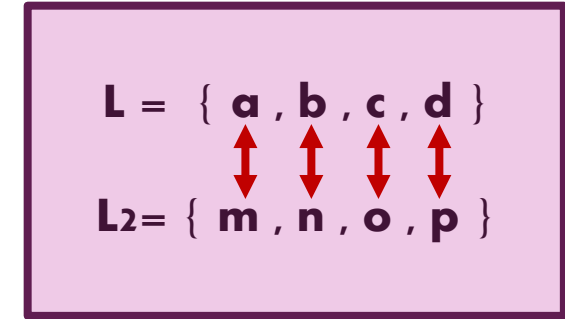
Task: Verify if it is mapping top of stack
 character, if so pop the stack



o
maps to
c

Stack

Decoding a Valid String



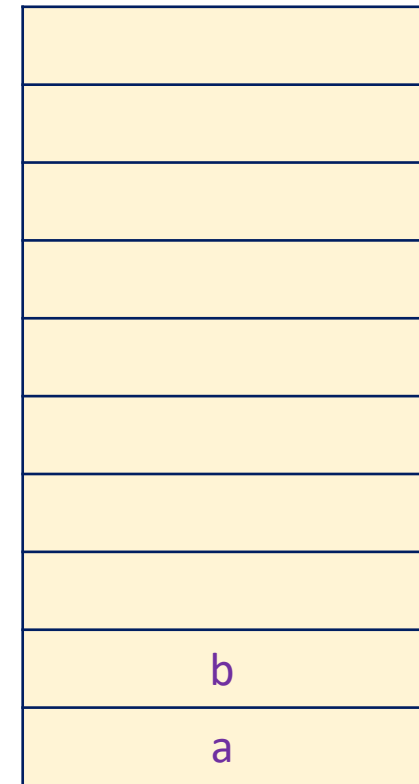
String: a b c g o n m

 ^

Current character: 0

Character Type: Belongs to ***L2***

Task: Verify if it is mapping top of stack character, if so **pop the stack**



Stack

Decoding a Valid String

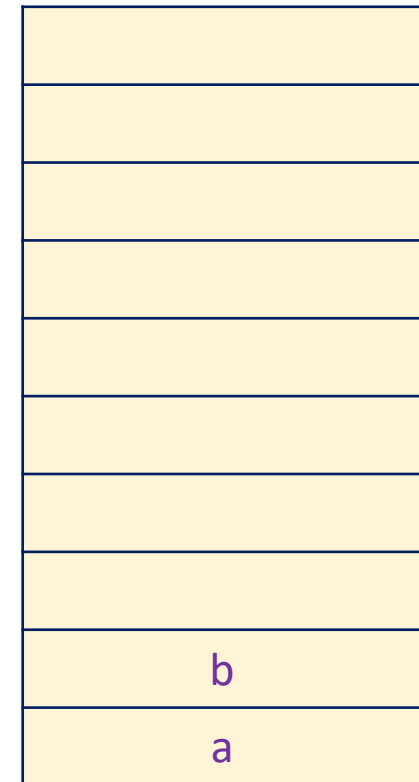
String: a b c g o n m

^

Current character: n

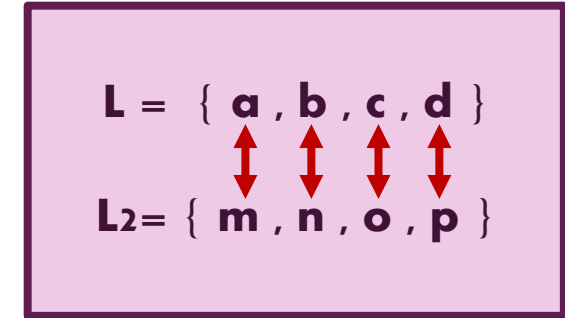
Character Type: Belongs to **L2**

Task: Verify if it is mapping top of stack
 character, if so pop the stack



Stack

Decoding a Valid String



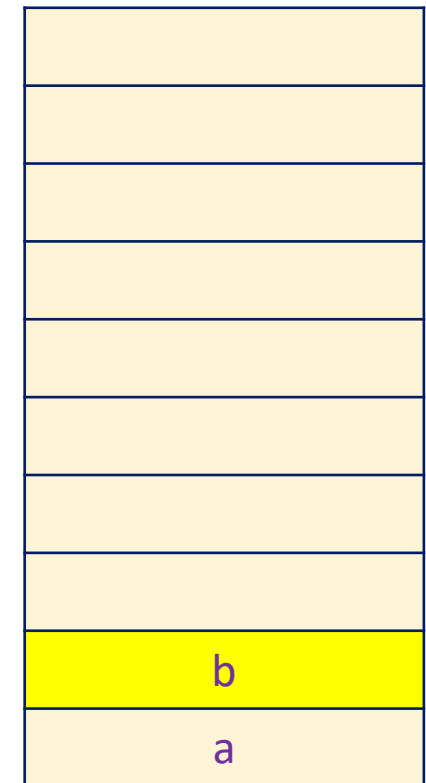
String: a b c g o n m

^

Current character: n

Character Type: Belongs to ***L2***

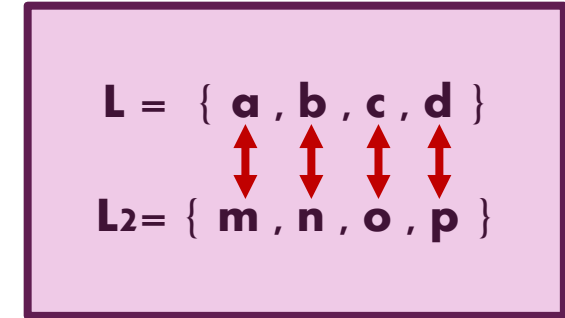
Task: Verify if it is mapping top of stack
 character, if so pop the stack



Stack

n
maps to
b

Decoding a Valid String



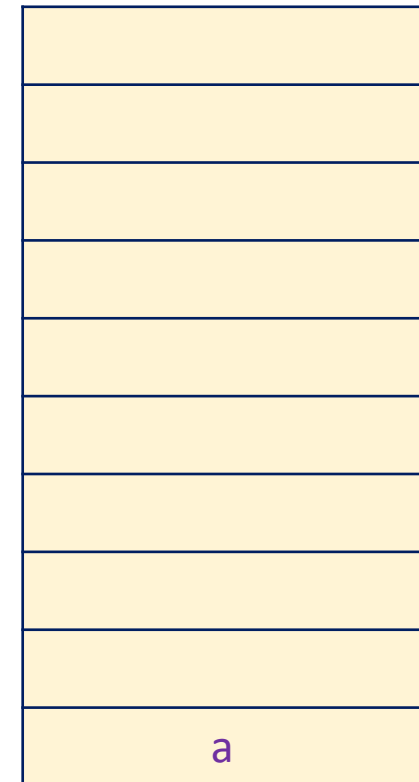
String: a b c g o n m

^

Current character: n

Character Type: Belongs to ***L2***

Task: Verify if it is mapping top of stack
character, if so **pop the stack**



Stack

Decoding a Valid String

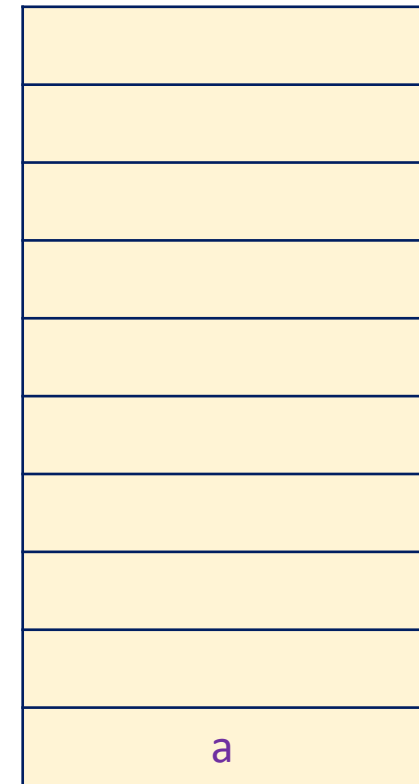
String: a b c g o n m

^

Current character: m

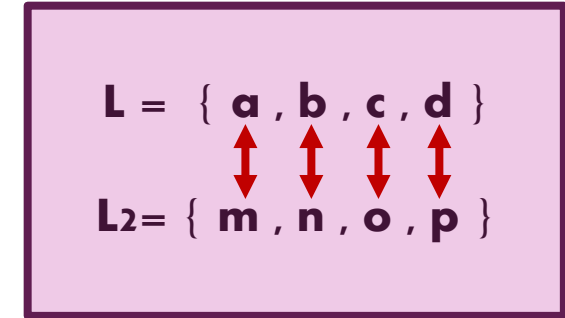
Character Type: Belongs to **L2**

Task: Verify if it is mapping top of stack
 character, if so pop the stack



Stack

Decoding a Valid String



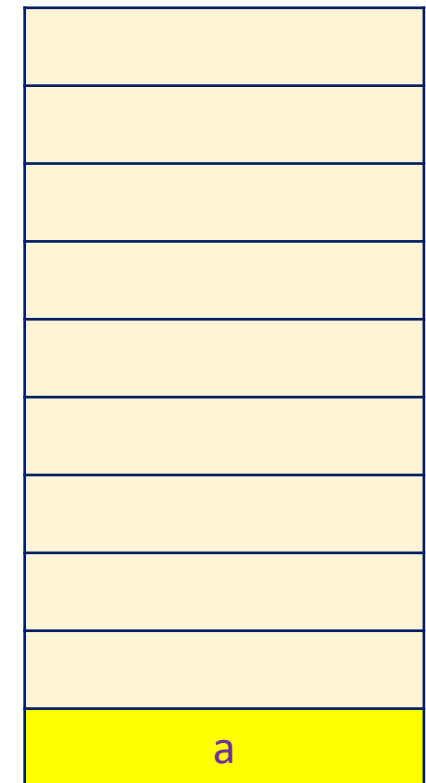
String: a b c g o n m

^

Current character: m

Character Type: Belongs to ***L2***

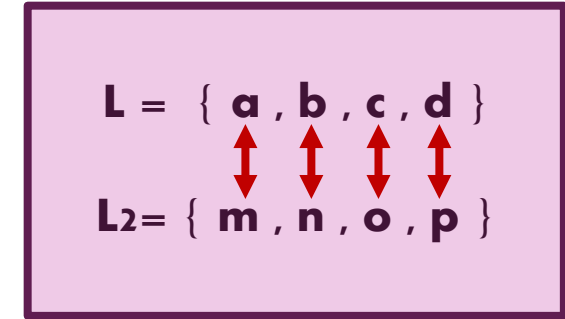
Task: Verify if it is mapping top of stack
 character, if so pop the stack



Stack

m
maps to
a

Decoding a Valid String



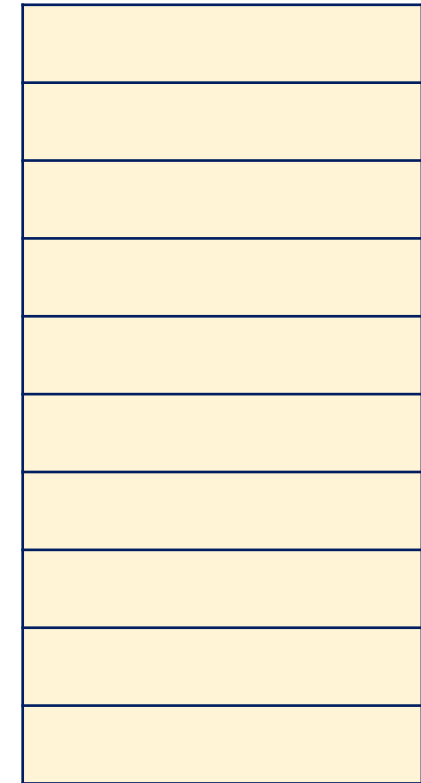
String: a b c g o n m

^

Current character: m

Character Type: Belongs to ***L2***

Task: Verify if it is mapping top of stack character, if so **pop the stack**



Stack

Decoding a Valid String

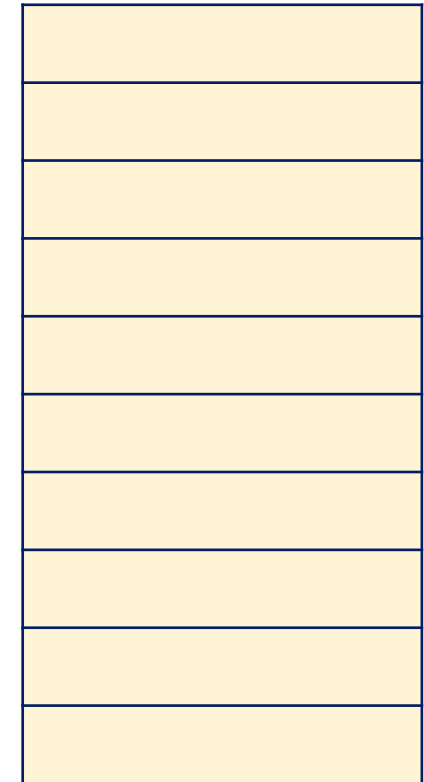
String: a b c g o n m

^

Current character: '/'

Character Type: End of string

Task: Verify stack is Empty



Stack

Decoding a Valid String

String: a b c g o n m

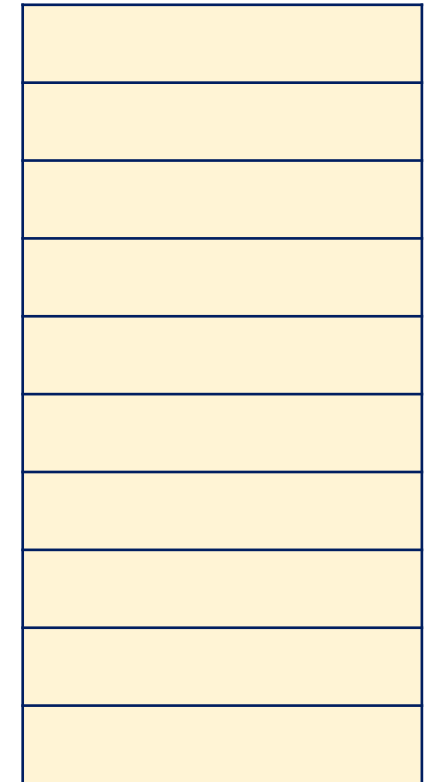
^

Current character: '/'

Character Type: End of string

Task: Verify stack is Empty

Stack is empty so our word is meaningful



Stack

Decoding a Valid String

String: a b c g o n m

Decoding Status: Meaningful

Next Step: Call RemoveExtraLetters ()



Removing Extra Letters

Input String: a b c g o n m

Output String: abc

Add the reduced string to your decoded message