CS 211 – Project 2

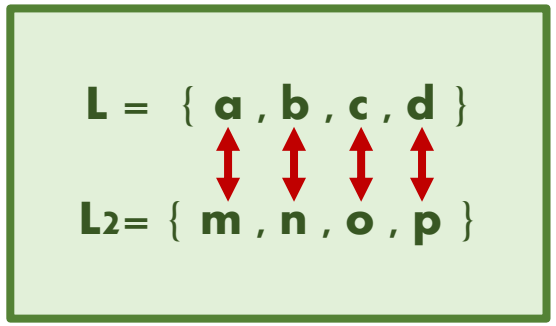
Discussion of Project

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}
  - L2={m, n, o, p}

- L2 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 encoding messages
- A string is meaningful if every existing letter from L is mapped/covered by corresponding letter from L2
- 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 ✓

dcayupom ? → **dcayupom** Invalid ✗

Decoding an Encrypted Message

Number of letters from L2 >= Number of letters from L

abcghonmpm ? → **abcghonmpm** Valid ✓

abcghon ? → **abcghon** Invalid ✗

Decoding an Encrypted Message

Letters from L should occur before letters from L2

abcghonm ? → **abcghonmpm** 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 Data Structure

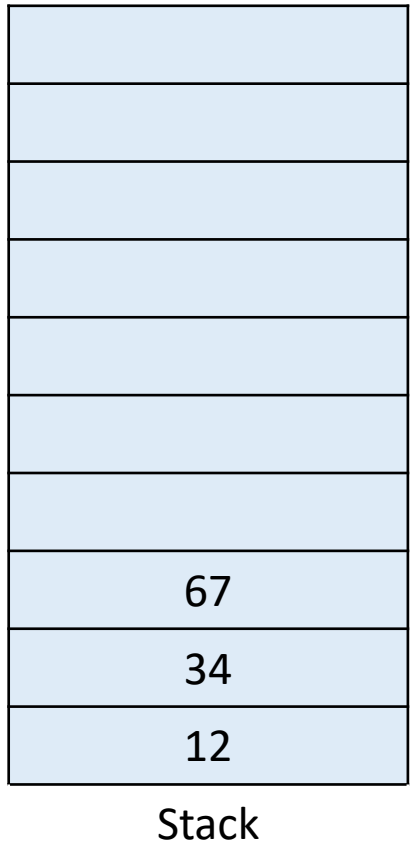
- Operations on stack
  - Push(12)

Stack Data Structure

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

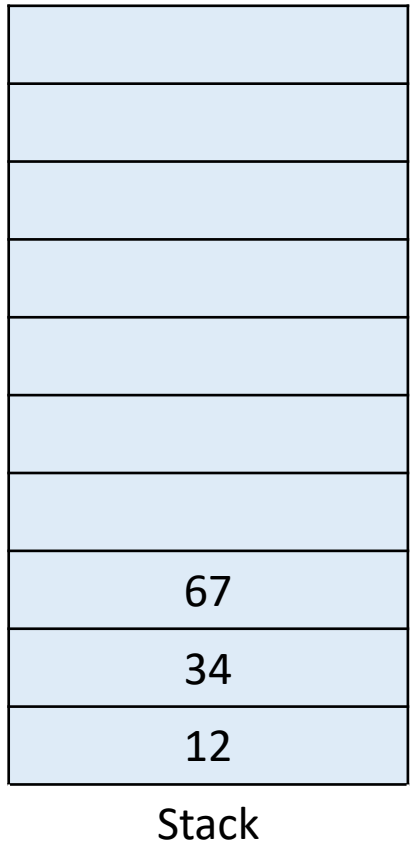
Stack Data Structure

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



Stack Data Structure

- Operations on stack
  - Push(12)
  - Push (34)
  - Push(67)
  - Top() → 67



Stack Data Structure

- Operations on stack
  - Push(12)
  - Push (34)
  - Push(67)
  - Top() → 67
  - Pop()

Stack Data Structure

- Operations on stack
  - Push(12)
  - Push (34)
  - Push(67)
  - Top() → 67
  - Pop()
  - Pop()

Stack Data Structure

- Operations on stack
  - Push(12)
  - Push (34)
  - Push(67)
  - Top() → 67
  - Pop()
  - Pop()
  - Top() → 12

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

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

Decoding a Valid String

String: a b c g o n m

^

Current character: a

Character Type: Belongs to **L**

Task: Push onto 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

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

Decoding a Valid String

String: a b c g o n m

^

Current character: c

Character Type: Belongs to **L**

Task: Push onto stack

Decoding a Valid String

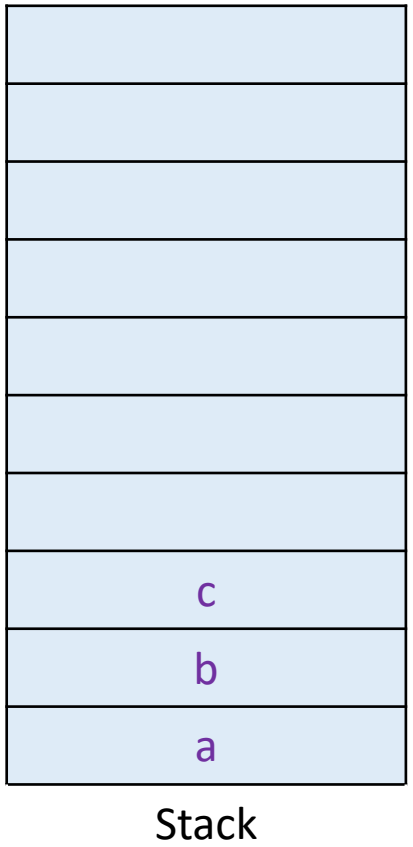
String: a b c g o n m

^

Current character: c

Character Type: Belongs to **L**

Task: Push onto stack



Decoding a Valid String

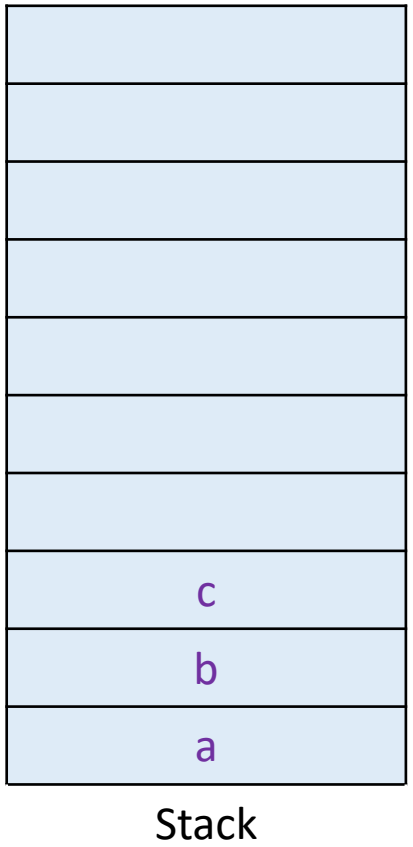
String: a b c g o n m

^

Current character: g

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

Task: ignore



Decoding a Valid String

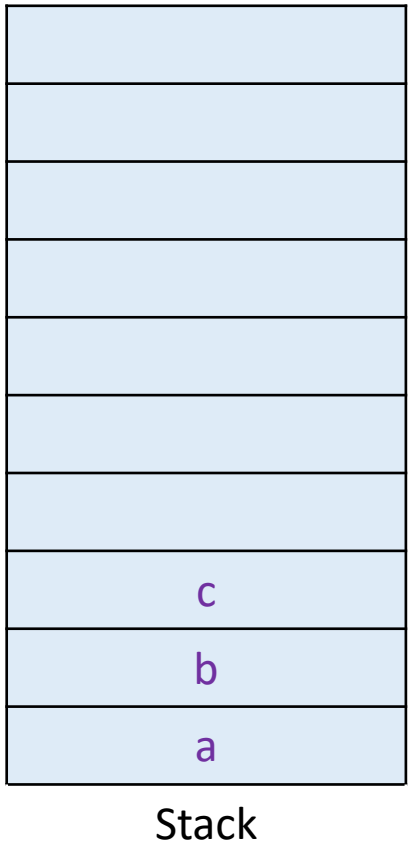
String: a b c g o n m

^

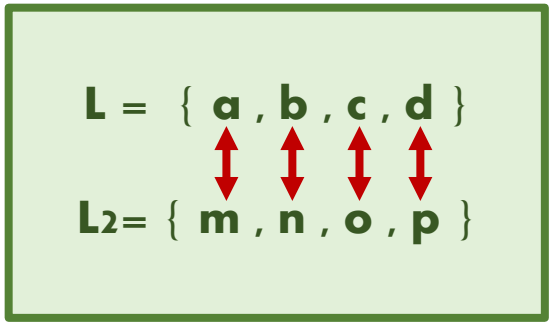
Current character: o

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

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



Decoding a Valid String



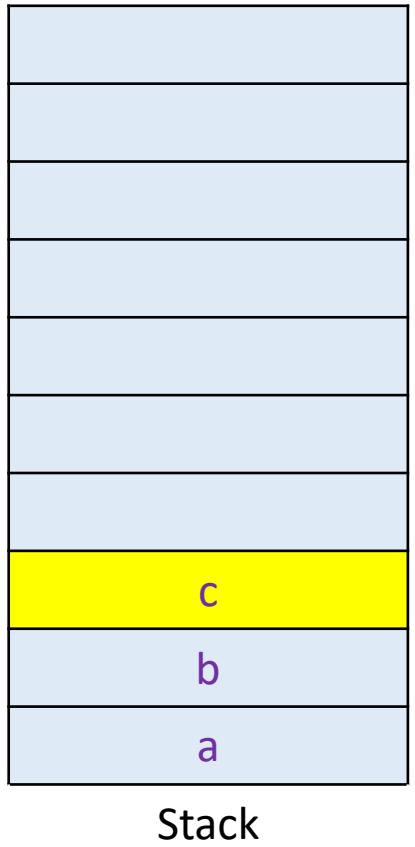
String: a b c g o n m

^

Current character: o

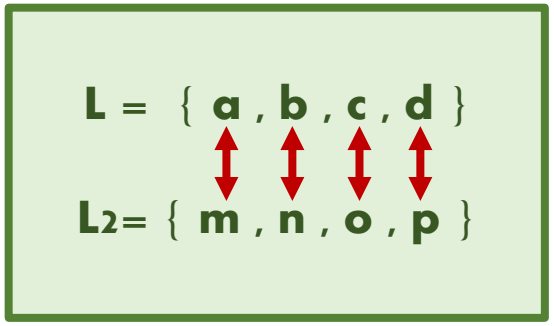
Character Type: Belongs to ***L2***

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



o maps to c

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: n

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

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

Decoding a Valid String

L = { a , b , c , d }
L2= { m , n , o , p }

String: a b c g o n m
^

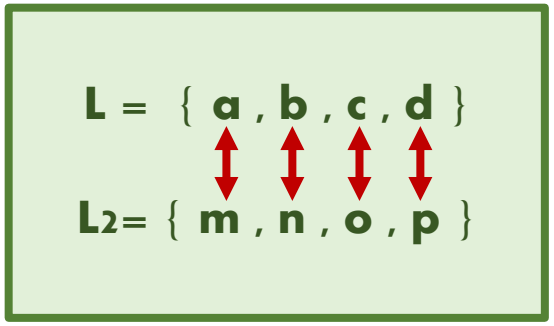
Current character: n

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

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

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

L = { a , b , c , d }
L2= { m , n , o , p }

String: a b c g o n m
^

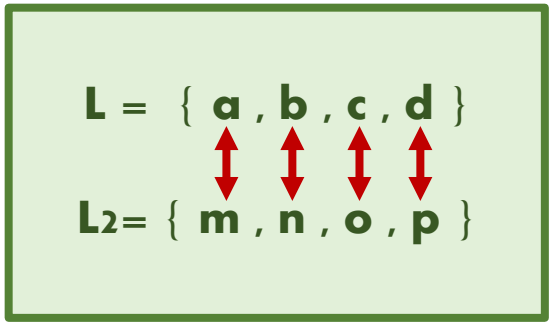
Current character: m

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

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

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: ‘/0’

Character Type: End of string

Task: Verify stack is Empty

Decoding a Valid String

String: a b c g o n m

^

Current character: ‘/0’

Character Type: End of string

Task: Verify stack is Empty

Stack is empty so our word is meaningful

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