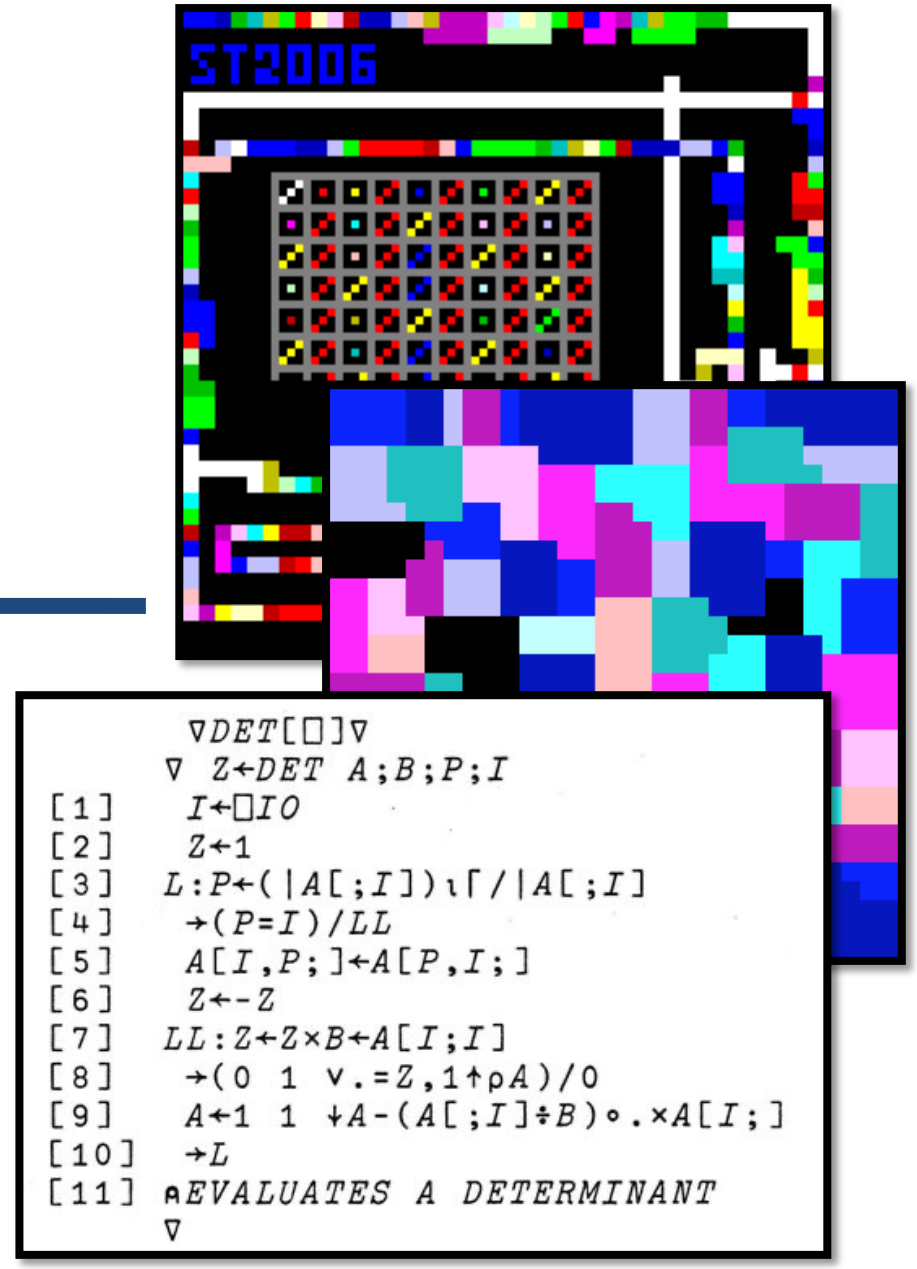
How Esoteric Can Programming Be?



Prachya Boonkwan

Language and Semantics Technology Lab (LST)
National Electronics and Computer Technology Center (NECTEC)

prachya.boonkwan@nectec.or.th, kaamanita@gmail.com

Slides can be found at https://tinyurl.com/y8rze72m

Who? Me?

- Nickname: **Arm** (P'/N'/E' Arm, etc.)
- Born: Aug 1981
- Work: researcher at NECTEC since 2005
- Education
  - B.Eng & M.Eng, CPE Kasetsart University
  - Obtained Ministry of Science Scholarship in early 2008
  - Did a PhD in Informatics (AI & Computational Linguistics) at University of Edinburgh, UK from 2008 to 2013 (4.5 years)

Princes Garden, Edinburgh

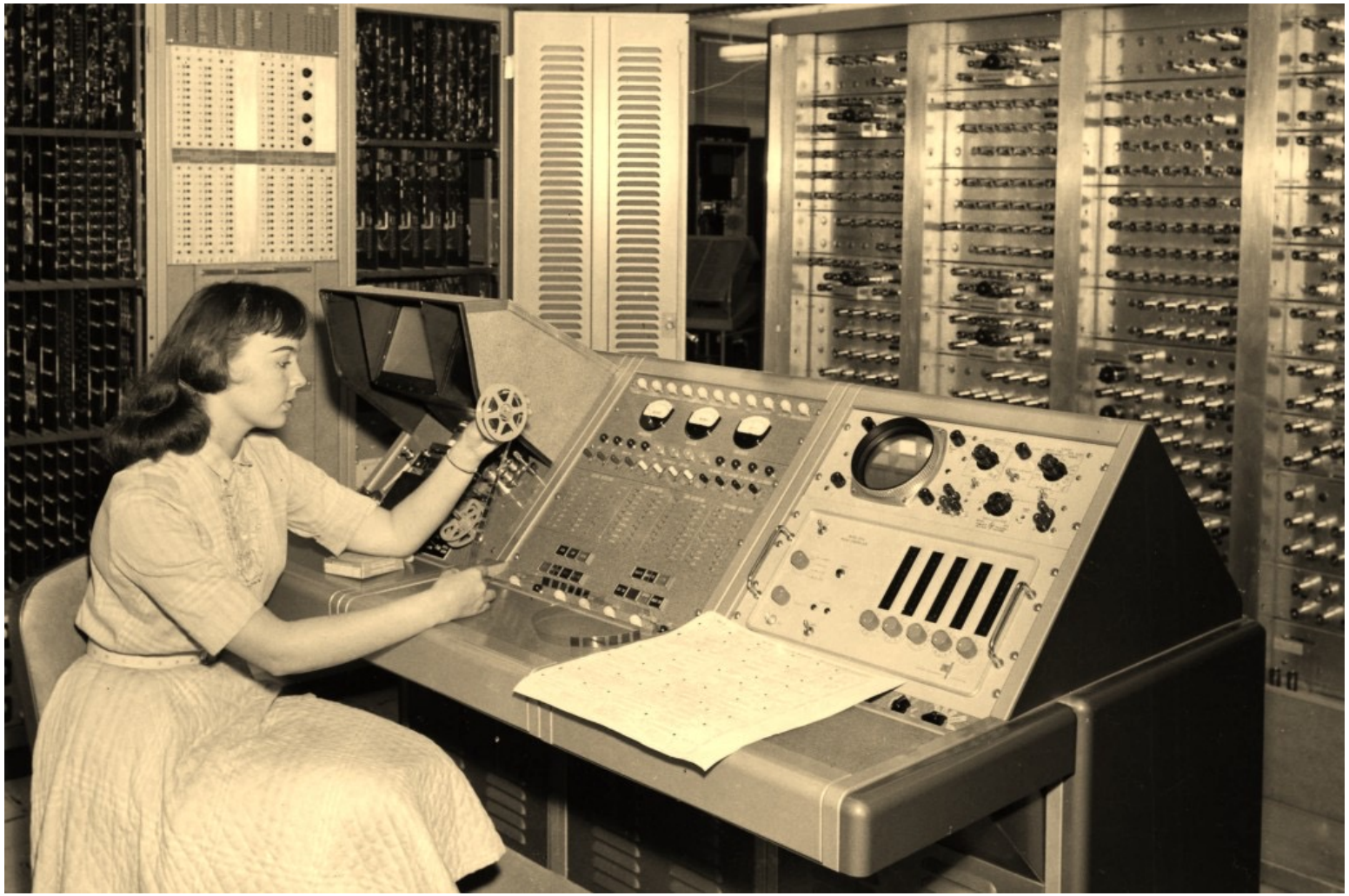
ช่วงเวลาแห่งการป้ายยามาถึงแล้ว

Outline

- Introduction
- Esoteric programming
  - Array-oriented languages
  - Turing-Machine-inspired languages
  - Jocular languages
  - Poetic languages
  - Pictorial languages
- Discussion
- Conclusion

1. Introduction

Timeline of Computer Programming



Eras	Programming methods
30s	hard wiring, machine code
40s	Assembly
50s	FORTRAN, RPG, LISP, COBOL
60s	APL, PL/I, SNOBOL, BASIC
70s	C, Prolog, ML, SQL, Smalltalk
80s	Ada, C++, Obj C, Erlang
90s	Python, Haskell, Java, JavaScript, Lua, Ocaml, R, Visual Basic
2000s	C#, Clojure, D, F#, Scala
2010s	Kotlin, Rust, Swift, Dart

Machine Code

- Univac I (1951)

Mem Cell	Left Instr	Right Instr	Comments
000	B00 100		Load data at addr [100] to Register A
		A00 101	Add data at addr [101] to the content of Register A
001	S00 102		Subtract the content of Register A with data at addr [102]
		H00 103	Store the content of Register A to the addr [103]
002	500 103		Print out data at addr [103]
		900 000	Halt

- Notion of 'byte' (8 bits) did not exist yet
  - 1 char = 6 bits
  - 1 word = 12 chars
  - Memory = 1000 words
- Programmer has to master 45 machine instructions (a.k.a. op-code)

Assembly Language

- Intel 8086 DOS Assembly (1978)

```
.model small
.stack 100h

.data
msg     db      'Hello world!$'

.code
start:
        mov     ah, 09h     ; Display the message
        lea     dx, msg
        int     21h
        mov     ax, 4C00h   ; Terminate the executable
        int     21h

end start
```

- The notion of ‘byte’ (8 bits) has existed since IBM360 (1964)
- Direct connection with the hardware
  - Registers & memory
  - Software and hardware interrupts
  - Hardware stack

```
      PROGRAM TPK
C     THE TPK ALGORITHM
C     FORTRAN-77 STYLE
      REAL A(0:10)
      READ(5,*) A
      DO 10 I = 10, 0, -1
         Y = FUN(A(I))
         IF (Y .LT. 400) THEN
            WRITE(6,9) I, Y
 9          FORMAT(I10. F12.6)
         ELSE
            WRITE(6,5) I
 5          FORMAT(I10, 'TOO LARGE')
         ENDIF
 10   CONTINUE
      END

      REAL FUNCTION FUN(T)
      REAL T
      FUN = SQRT(ABS(T)) + 5.0 * T**3
      END
```

Procedural Programming

- FORTRAN 77 (1977)
  - Human-readable code + math exprs.
  - Conditions and loops
  - Functions and subroutines
  - Formatted inputs and outputs

Declarative Programming

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :-
    parent_child(Z, X),
    parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

```
?- sibling(sally, erica).
 Yes
```

- Prolog (1972)
  - No explicit loops
  - Fact: relational database
  - Rule: logical exprs. = database query
  - Branching via cut ‘!’ (no backtracking)

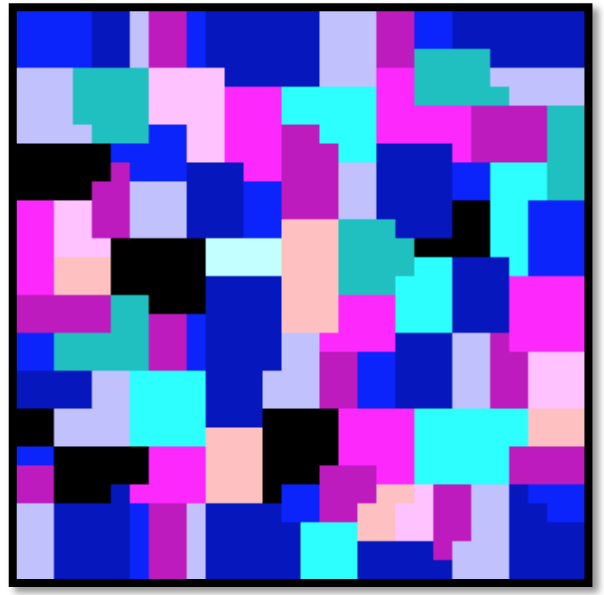

There is 1 Impostor among us

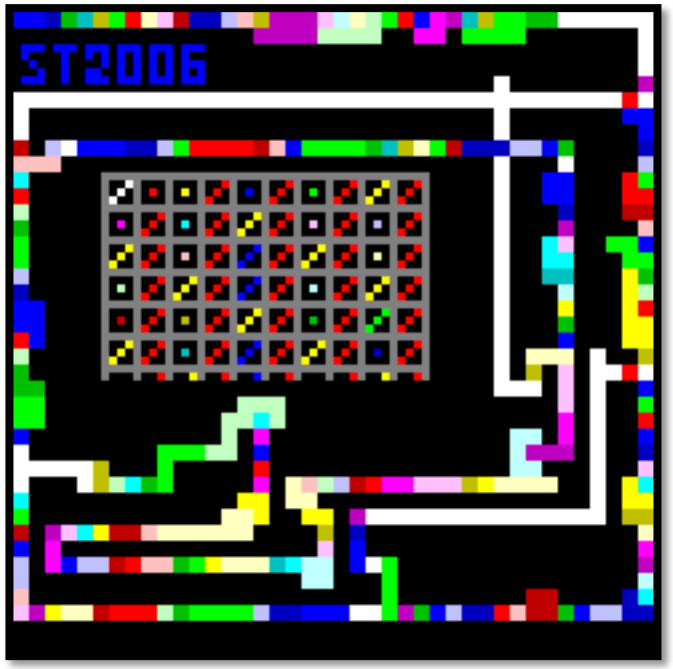
Esoteric Programming

- **esoteric** /ˌɛsəˈtɛrɪk ˌเอ-เสอะ-ˈเท-ริก/
  - (*adj.*) intended for or likely to be understood by only a small number of people with a specialized knowledge or interest [Oxford Dict.]
- **esoteric programming language (esolang)**
  - (*n.*) a programming language designed to **test the boundaries of computer programming language design**, as a proof of concept, as software art, as a hacking interface to another language (particularly functional programming or procedural programming languages), or as a joke [Wikipedia]

Esoteric Programming

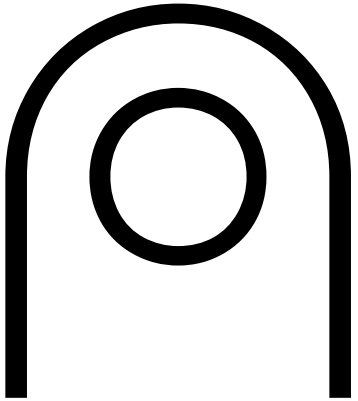
- Array-oriented languages
- Turing-Machine-inspired languages
- Jocular languages
- Poetic languages
- Pictorial languages






```
      ∇DET[⎕]∇
     ∇ Z←DET A;B;P;I
[1]    I←⎕IO
[2]    Z←1
[3]   L:P←(|A[;I])⍳⌈/|A[;I]
[4]    →(P=I)/LL
[5]    A[I,P;]←A[P,I;]
[6]    Z←-Z
[7]   LL:Z←Z×B←A[I;I]
[8]    →(0 1 ∨.=Z,1↑⍴A)/0
[9]    A←1 1 ↓A-(A[;I]÷B)∘.×A[I;]
[10]   →L
[11]  ⍝EVALUATES A DETERMINANT
      ∇
```

2. Esoteric Programming



2.1 Array-Oriented Languages

APL (A Programming Language)

- History
  - Developed by Kenneth E. Iverson in 1966
  - Very popular until late 1980s
  - Can be used as matrix calculator, spreadsheet, and data analysis tool
- Features
  - Single data type: multidimensional array
  - Using a large range of graphical symbols
  - Very concise code, easy to recite
  - Parallelized matrix computation

```
(~R∈R∘.×R)/R←1↓⍳N
```

This code lists up all prime numbers in $[1, N]$

Wanna give it a shot?
Go to https://tryapl.org
for interactive APL interpreter

```
(~R∊R∘.×R)/R←1↓⍳N
```

How Does That Work?

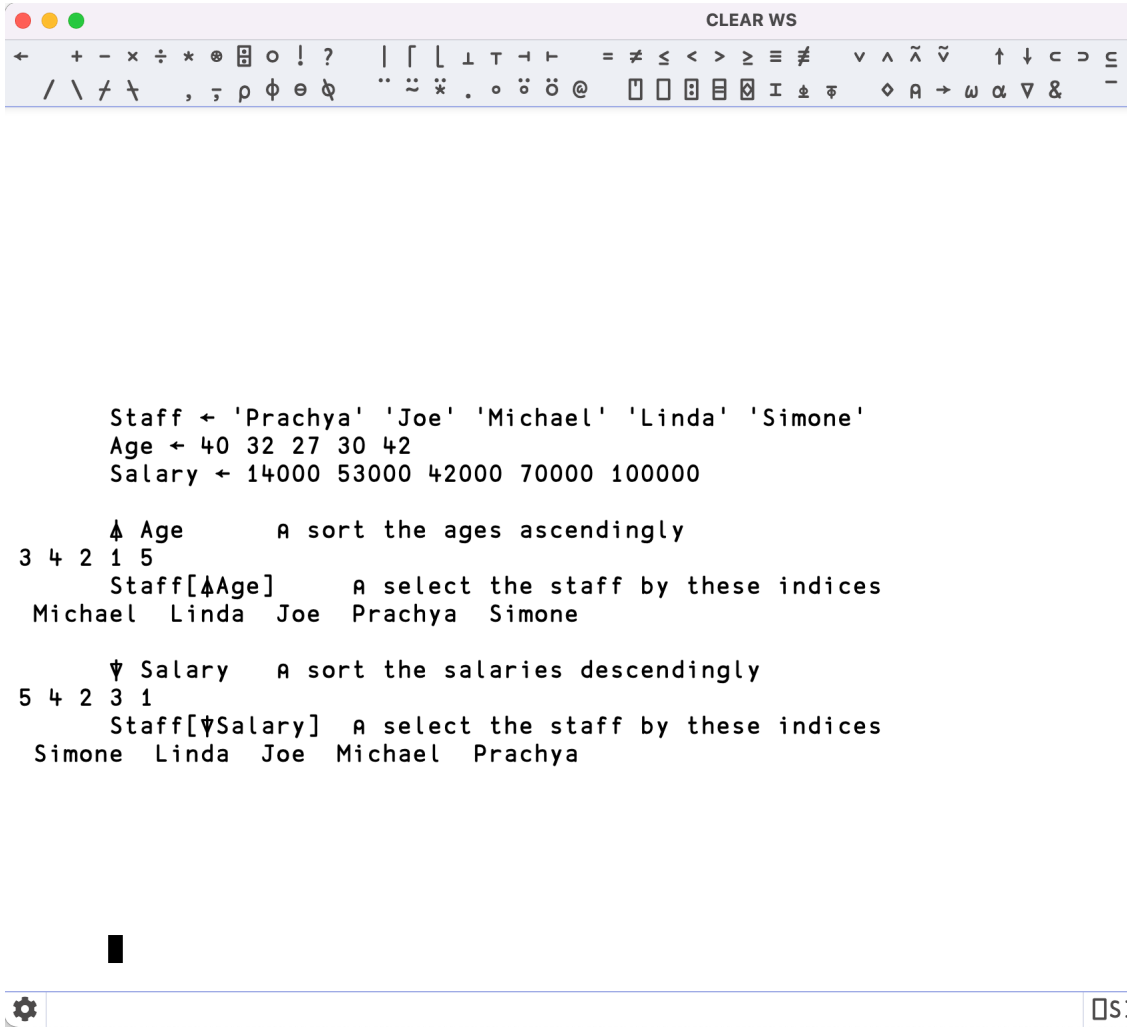
CLEAR WS

```
      N ← 10                  ⍝ set N to 10
      N
10
      ⍳N                      ⍝ create a list [1..N]
1 2 3 4 5 6 7 8 9 10

      1↓⍳N                    ⍝ drop column 1 out
2 3 4 5 6 7 8 9 10
      R ← 1↓⍳N                ⍝ let's save the result here

      R ∘.× R                 ⍝ outer product of R with ×
 4  6  8 10 12 14 16 18  20
 6  9 12 15 18 21 24 27  30
 8 12 16 20 24 28 32 36  40
10 15 20 25 30 35 40 45  50
12 18 24 30 36 42 48 54  60
14 21 28 35 42 49 56 63  70
16 24 32 40 48 56 64 72  80
18 27 36 45 54 63 72 81  90
20 30 40 50 60 70 80 90 100

      R ∊ R∘.×R               ⍝ check if any element of R is in this outer product
0 0 1 0 1 0 1 1 1

      ~ R∊R∘.×R               ⍝ now we look for non-member elements of R
1 1 0 1 0 1 0 0 0

      (~R∊R∘.×R) / R          ⍝ use these indices to select the elements of R
2 3 5 7
```

SI: 0 &: 1 Ln 2038, Col 6

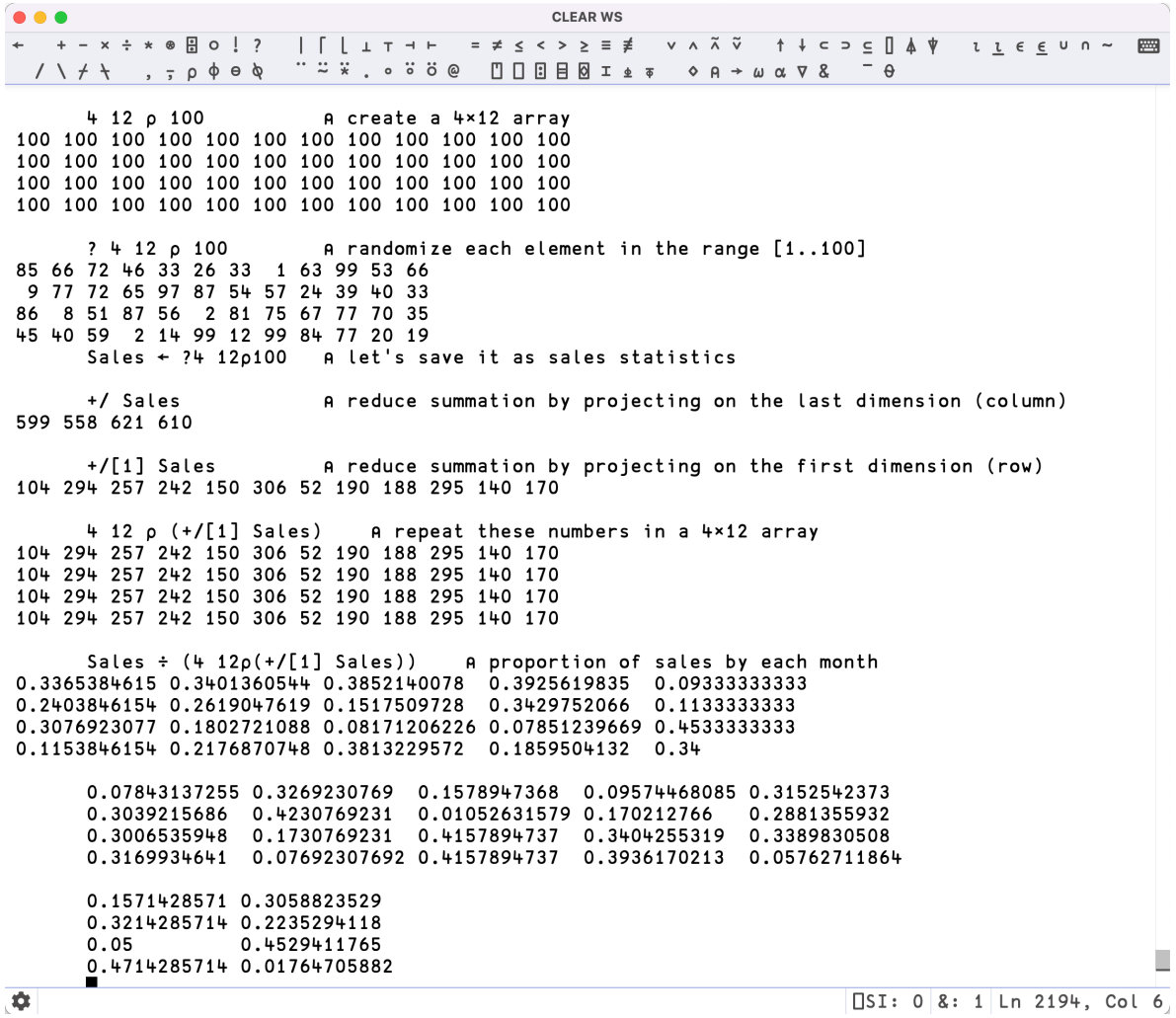
- APL interprets the expr from right to left
- Three types of operators
  - Monadic (RHS arg)
  - Dyadic (LHS & RHS args)
  - Adverb (modifying func.)
- Operator trains, e.g.
  - $f.g$ = inner product
  - $\circ.f$ = outer product of f
  - $f/$ = reduce
  - $f\ddot{}$ = each

Why was it so popular until late 1980s?



- APL can be used as a data analysis tool before the age of spreadsheet
- Many legacy libraries in finance are still in APL
- There were even APL conferences since 1969 to 2010, then it became Dyalog User Meetings ever since

Why was it so popular until late 1980s?



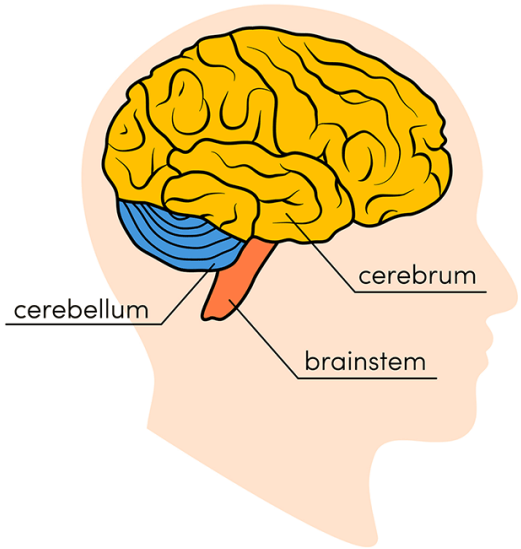
- APL can be used as a data analysis tool before the age of spreadsheet
- Many legacy libraries in finance are still in APL
- There were even APL conferences since 1969 to 2010, then it became Dyalog User Meetings ever since

APL makes a comeback as a hobby

- It is array-oriented and suitable for GPU-based machine learning
  - You can leverage mac GPUs, too! (via Dyalog APL and ArrayFire)
- In quantum computing, we have to think in matrix multiplication, complex numbers, and array manipulation
- The code is generally concise and easy to maintain as it always fits in one page

Dyalog delivers an APL-based development environment that allows both subject matter experts and IT specialists to efficiently convert ideas into software solutions.

https://www.dyalog.com



2.2 Turing-Machine-Inspired Languages

Brainfuck

- Created by Urban Müller in 1993
- Requiring us to break down commands into microscopic steps

Command	Description
+	Increase the data at the pointer by 1
–	Decrease the data at the pointer by 1
>	Shift the pointer to the right
<	Shift the pointer to the left
.	Print the data at the pointer
,	Read the data from the keyboard and store it at the pointer
[	If the data at the pointer is 0, jump to the matching]
]	If the data at the pointer is not 0, jump to the matching [

```
++>+++++[<+>-]++++++++
[<++++++>-]<.
```

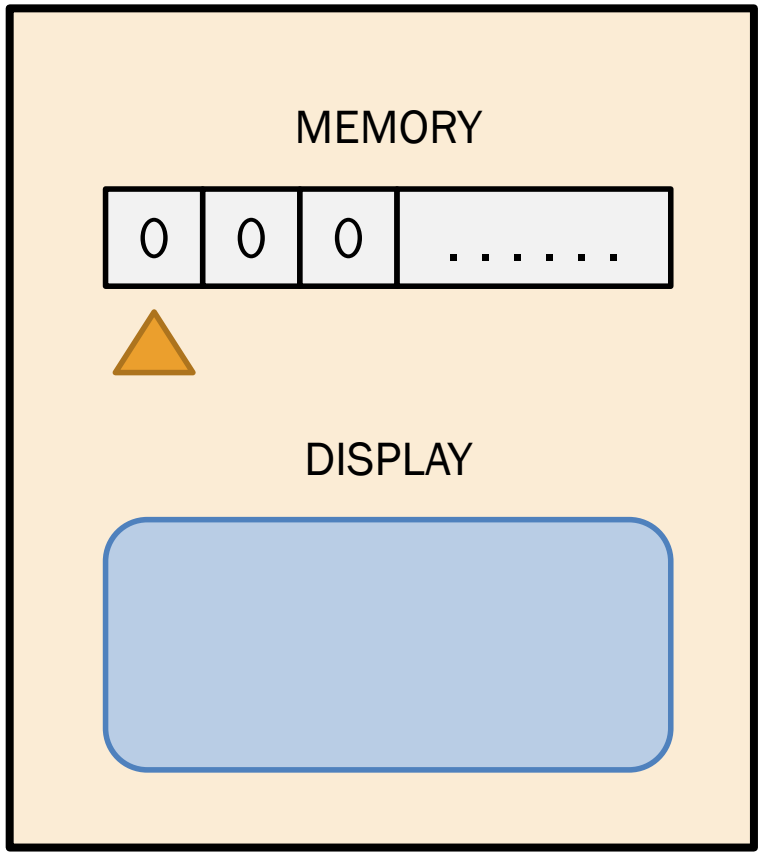
This code computes 2+5 and print the result to the screen

Wanna give it a shot? Go to http://www.bf.doleczek.pl for interactive Brainfuck

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

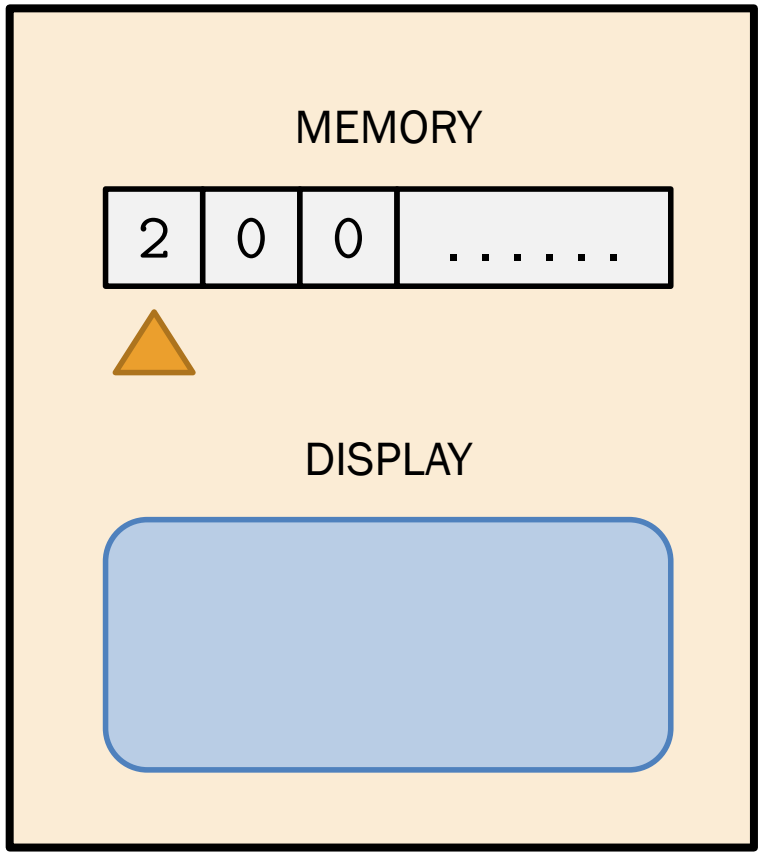


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

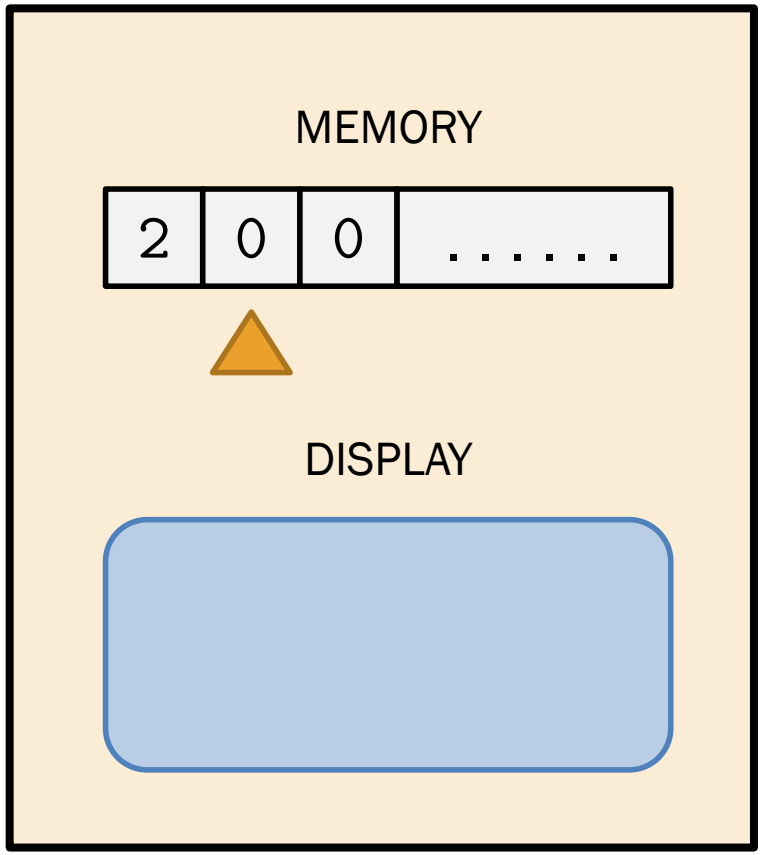


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

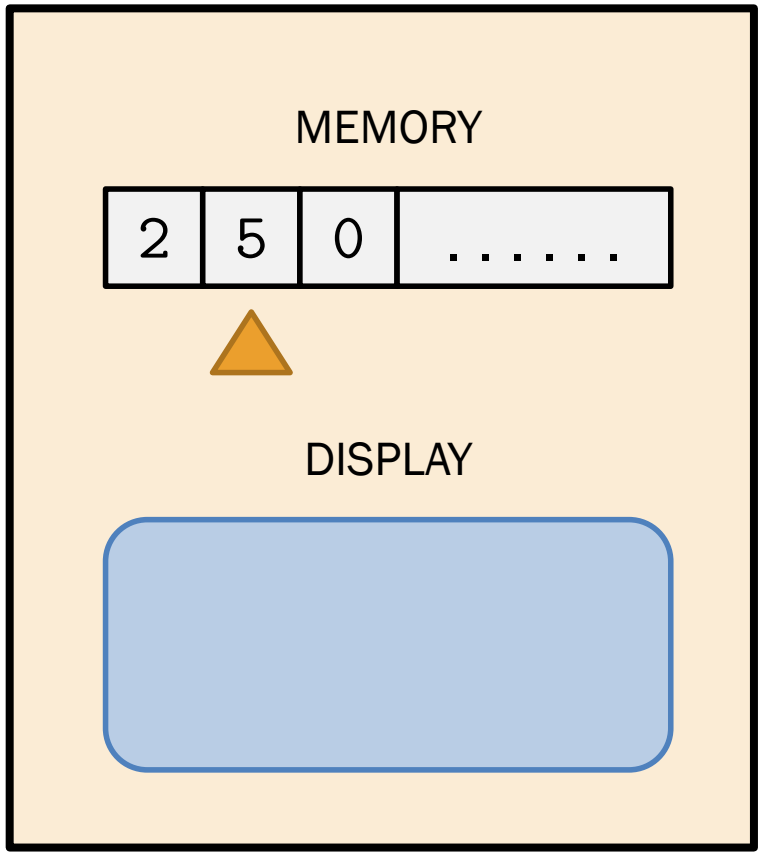


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

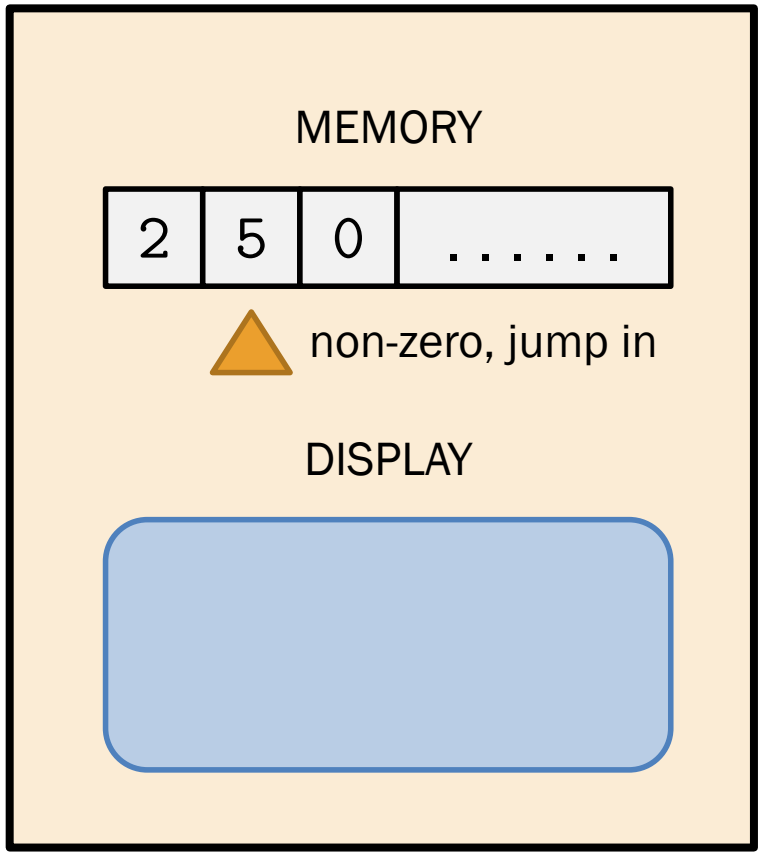


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

      < +

      > -

]

++++++++

[<++++++>-]<.
```

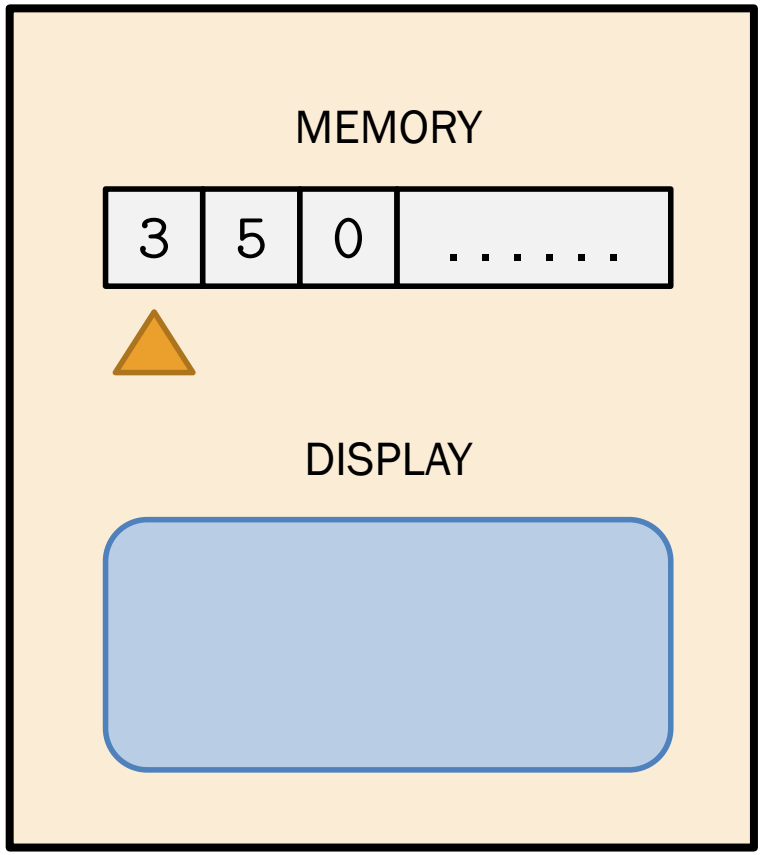


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

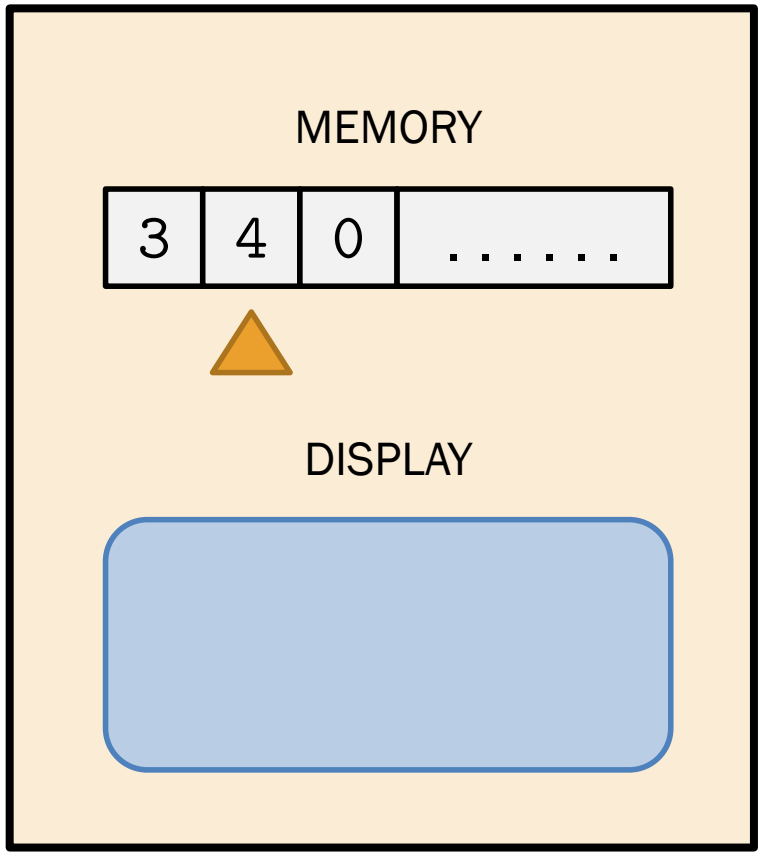


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

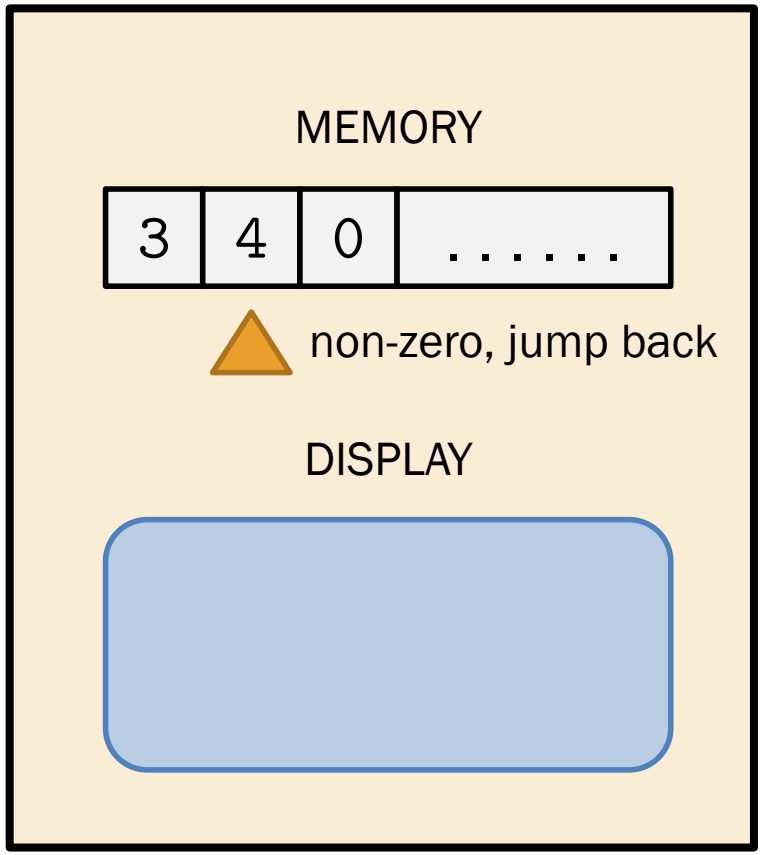


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    < +

    > -

]

++++++++

[<++++++>-]<.
```

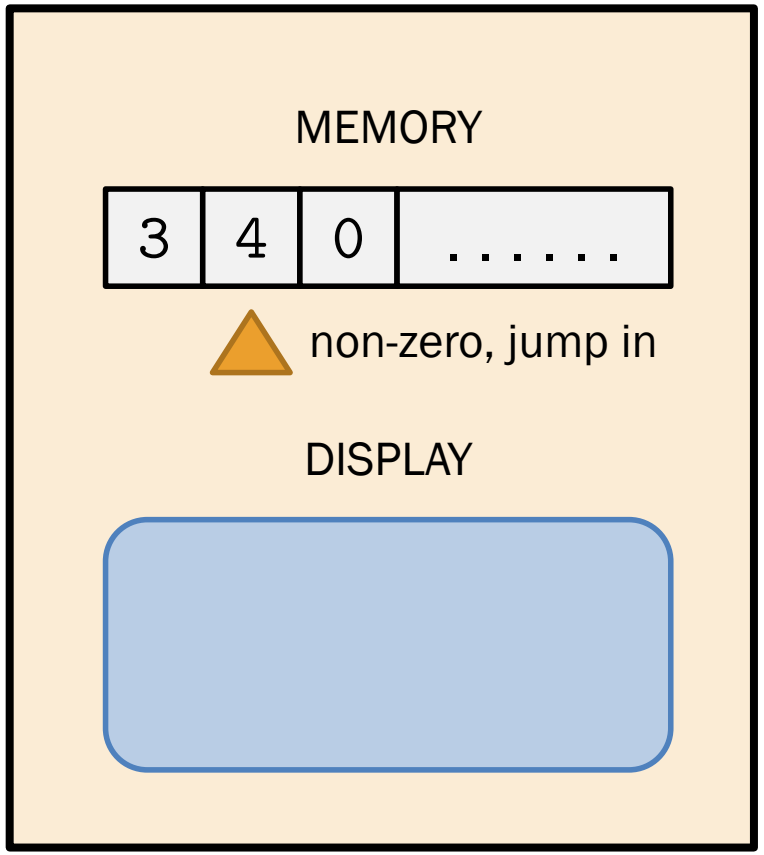


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

      < +

      > -

]

++++++++

[<++++++>-]<.
```

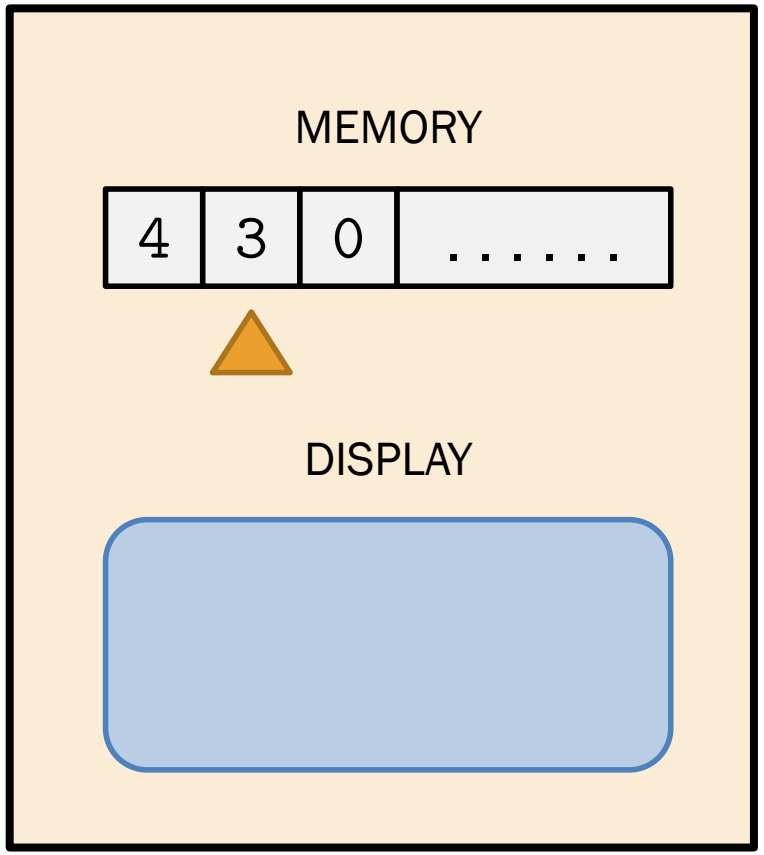


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

      <+ >-

]

++++++++

[<++++++>-]<.
```

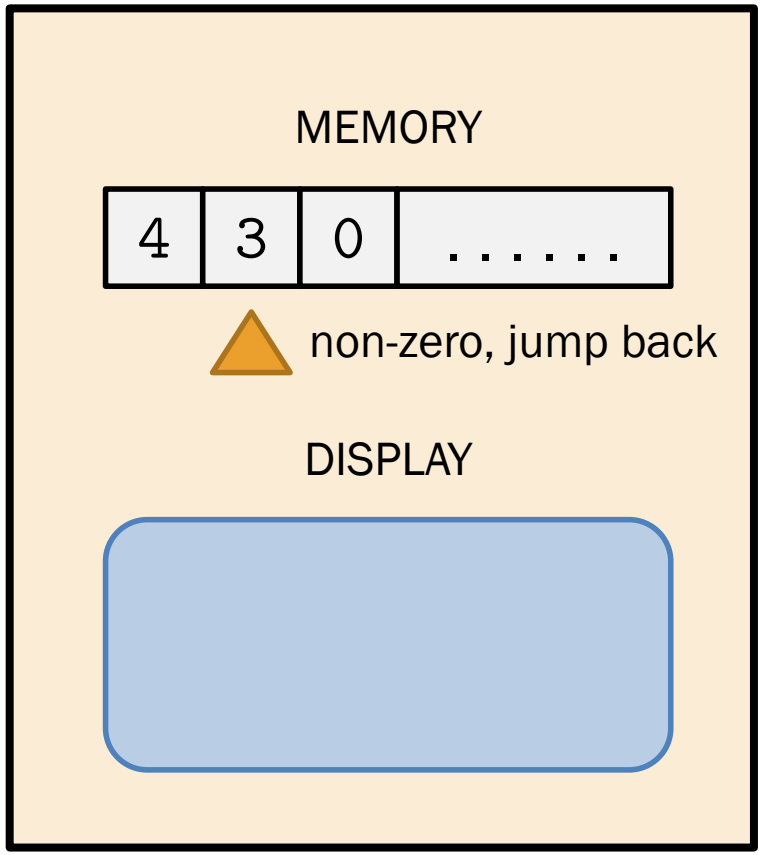


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    <+ >-

]

++++++++

[<++++++>-]<.
```

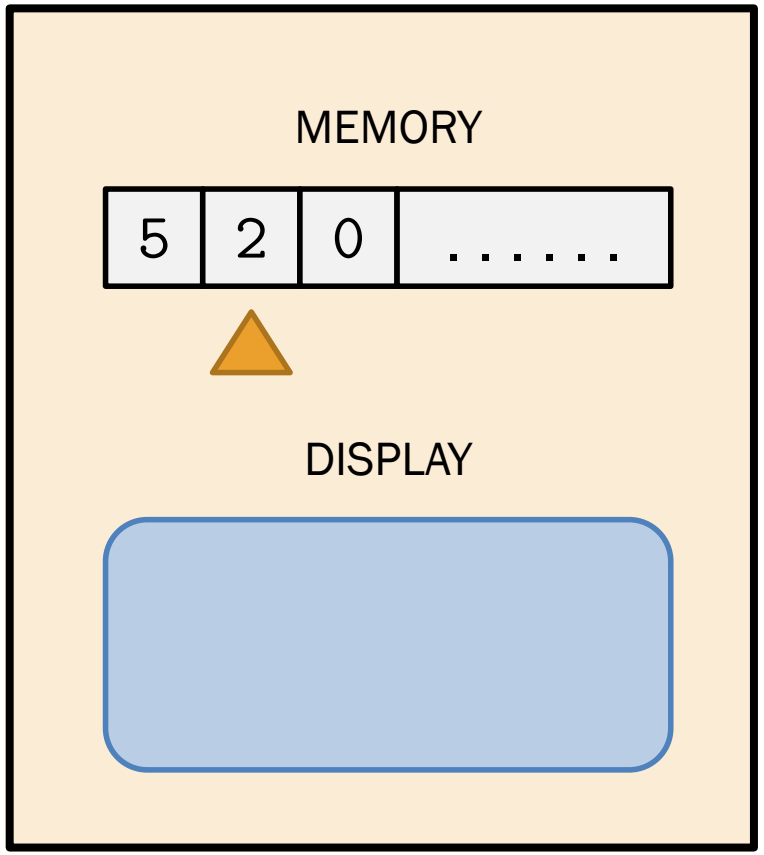


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

      <+ >-

]

++++++++

[<++++++>-]<.
```

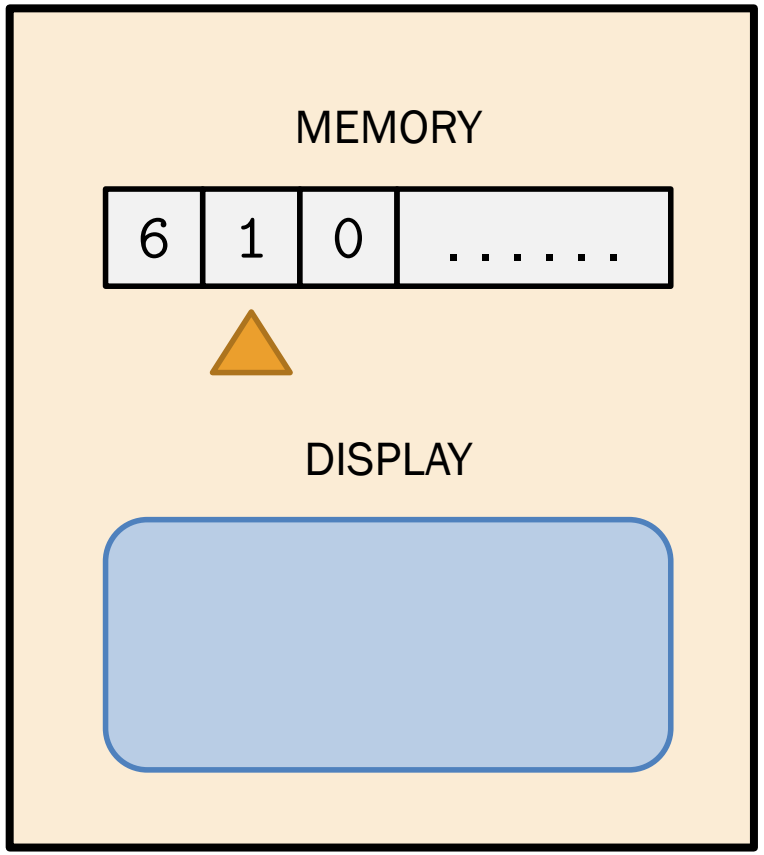


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    <+ >-

]

++++++++

[<++++++>-]<.
```

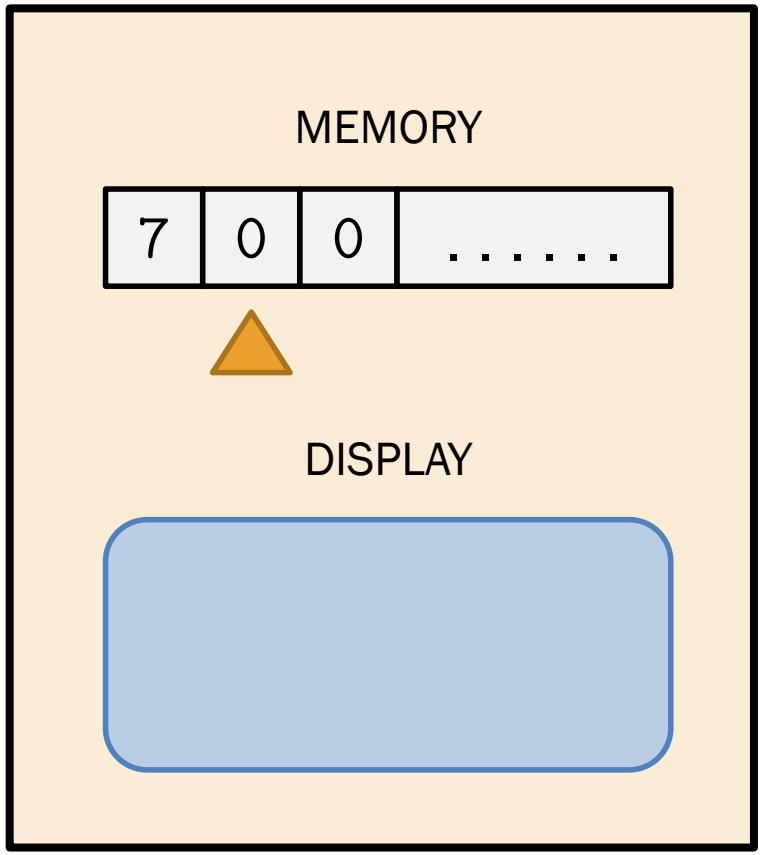


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    <+ >-

]

++++++++

[<++++++>-]<.
```

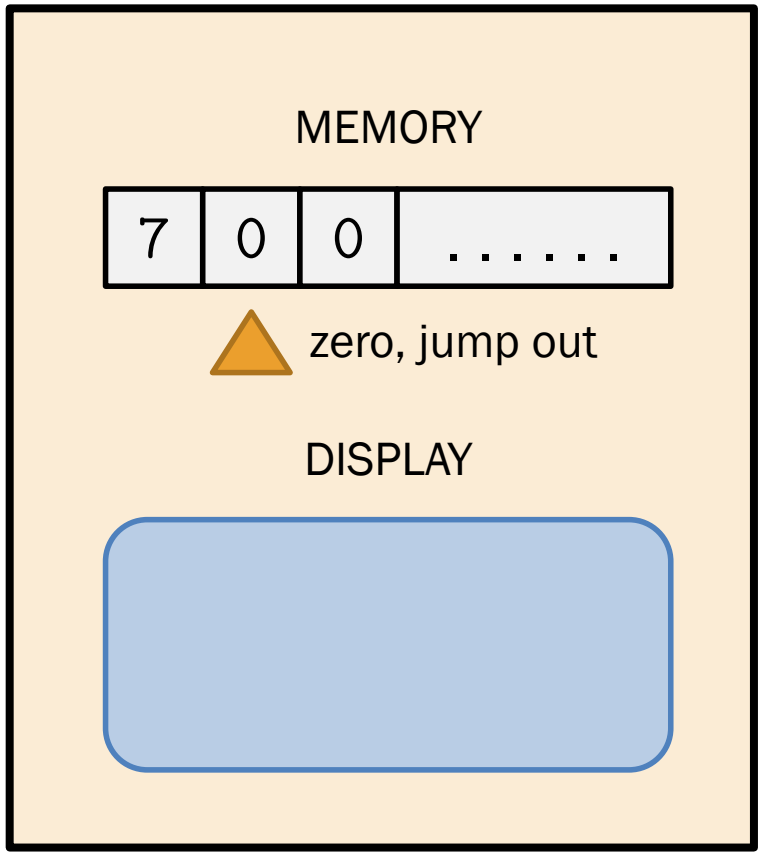


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    <+ >-

]

++++++++

[<++++++>-]<.
```

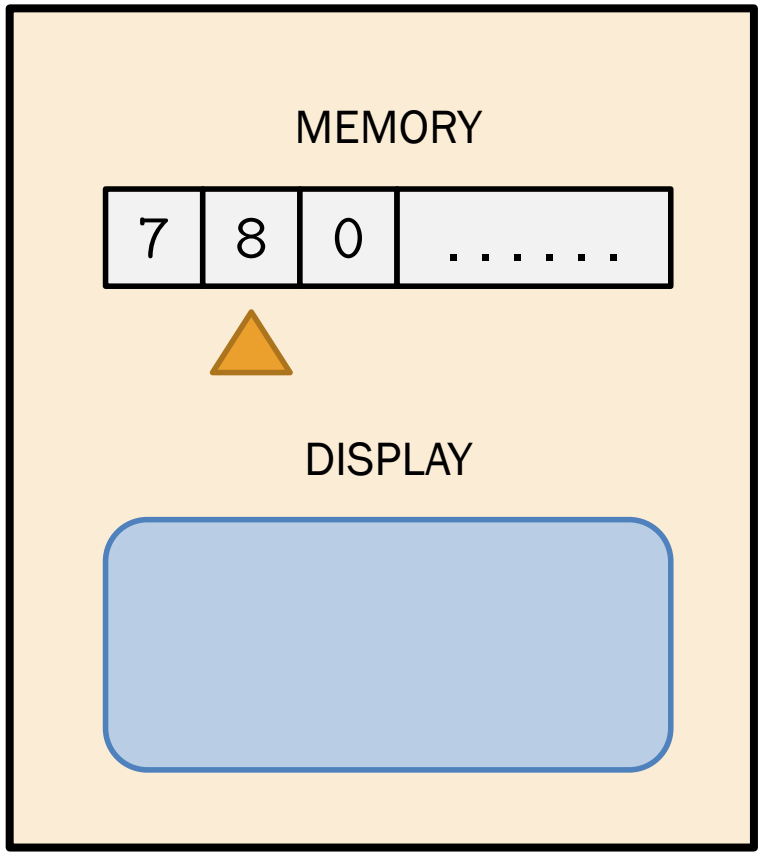


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++

> +++++

[

    <+ >-

]

++++++++

[<++++++>-]<.
```

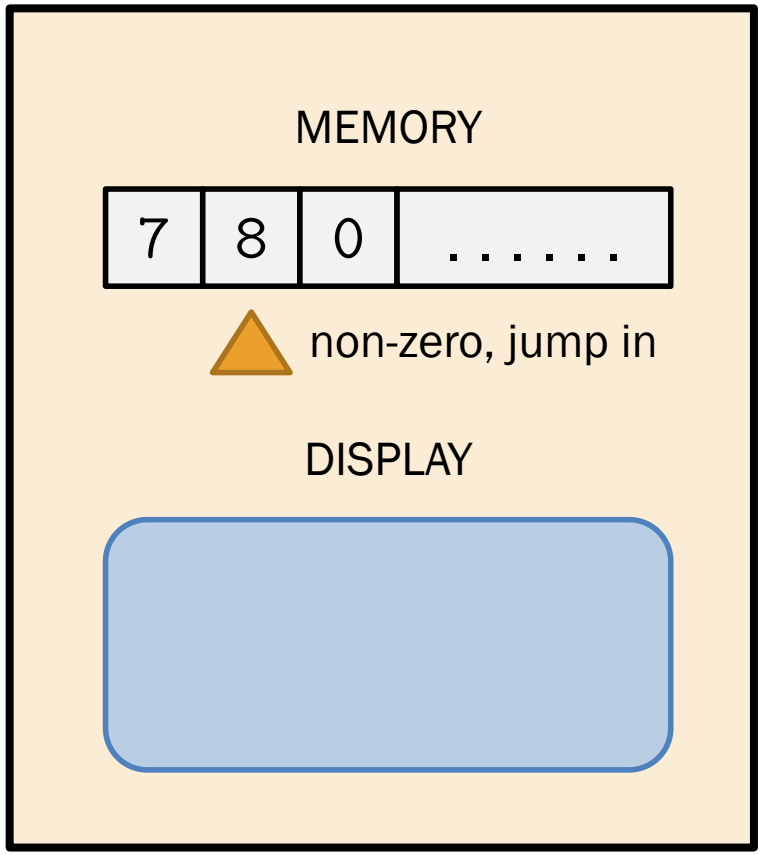


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

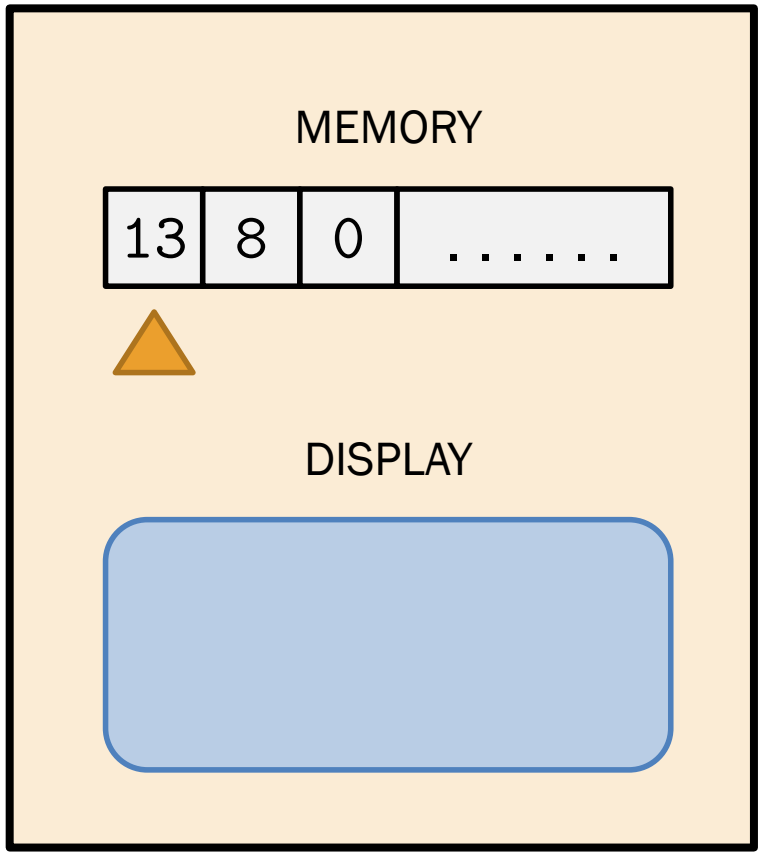


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

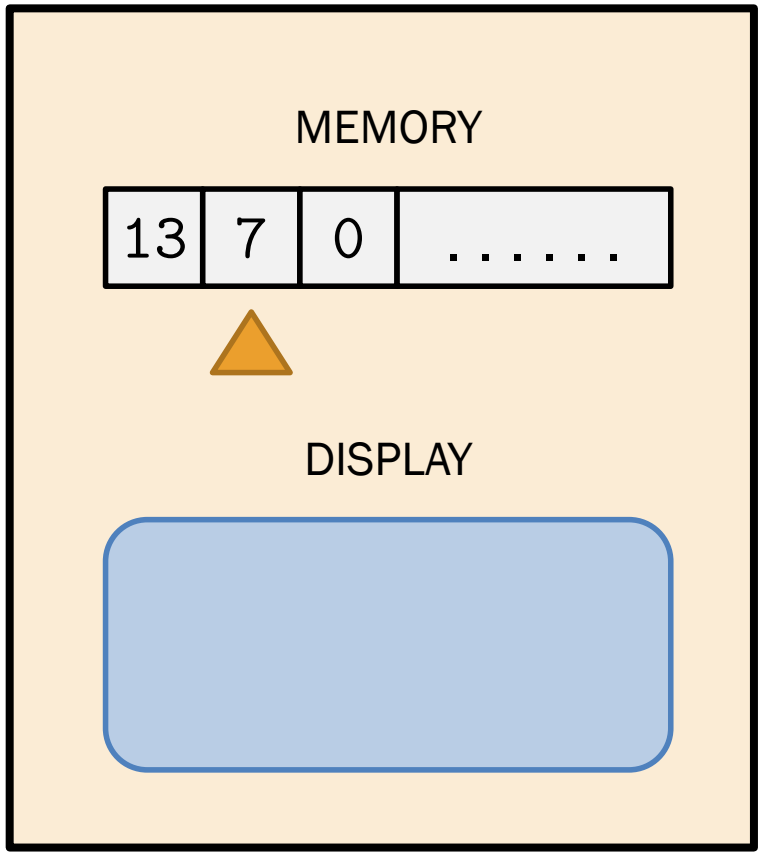


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

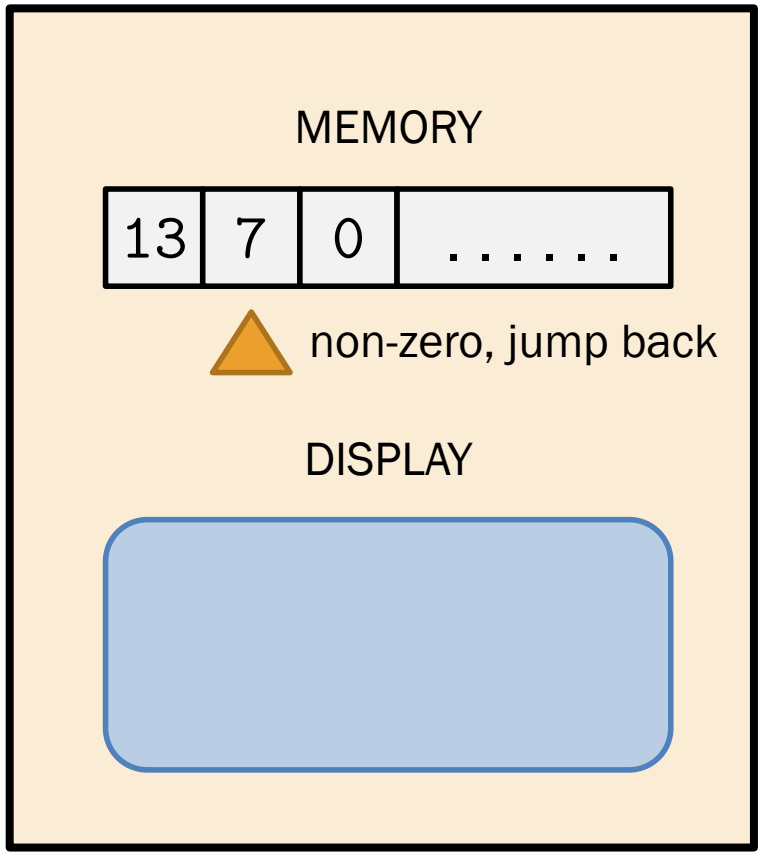


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

      < ++++++

      > -

]

<

.
```

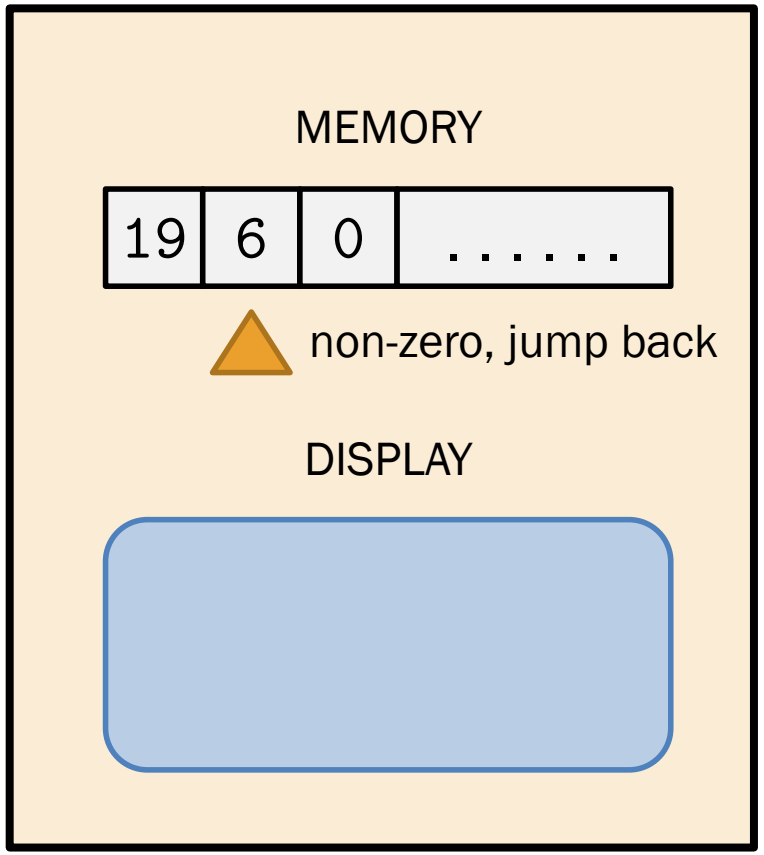


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

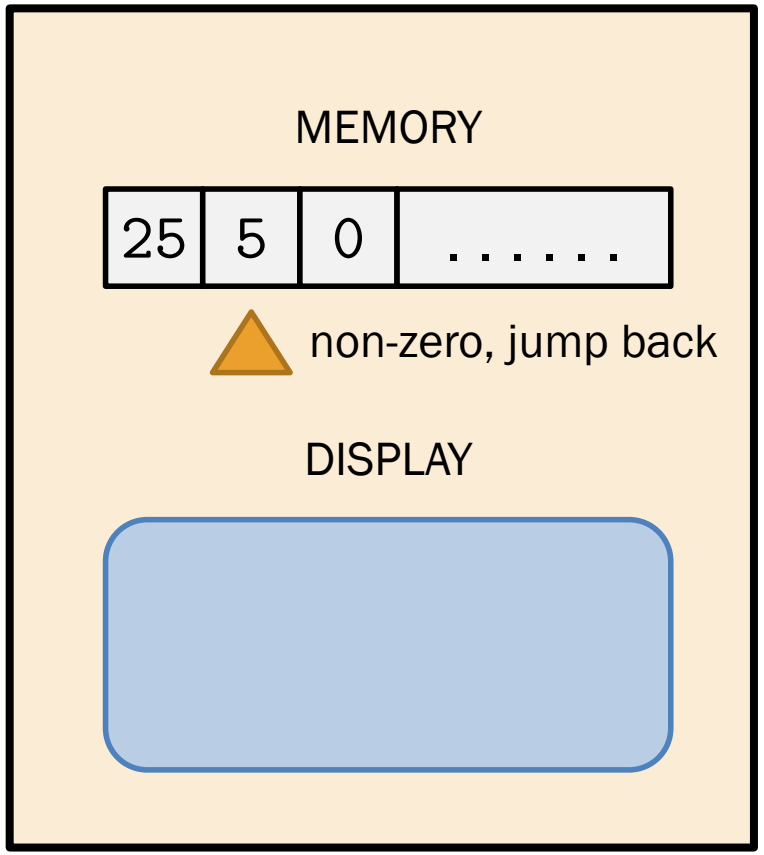


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

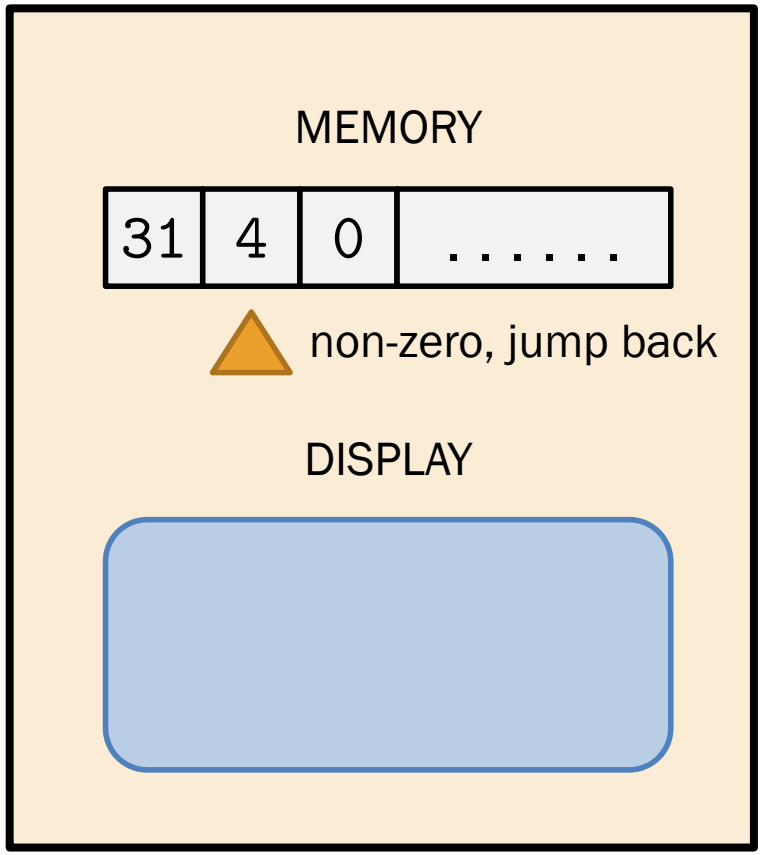


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

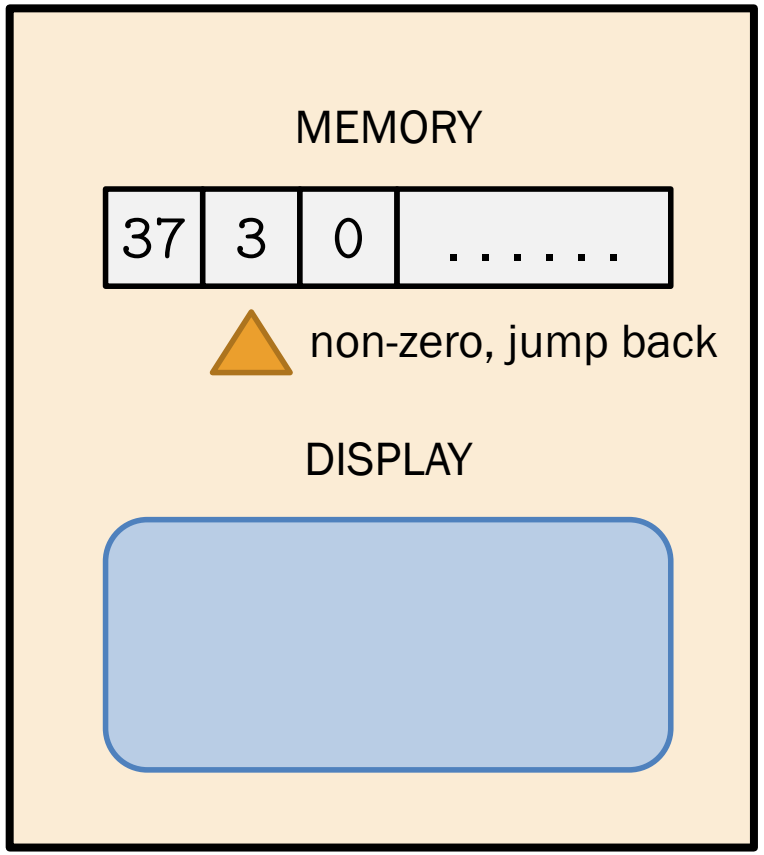


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

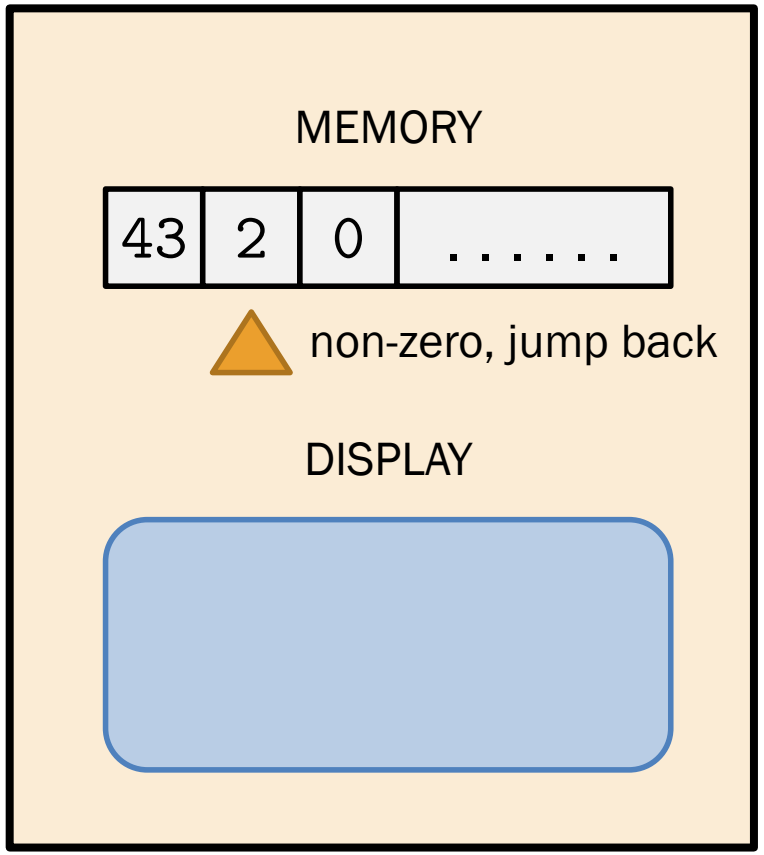


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

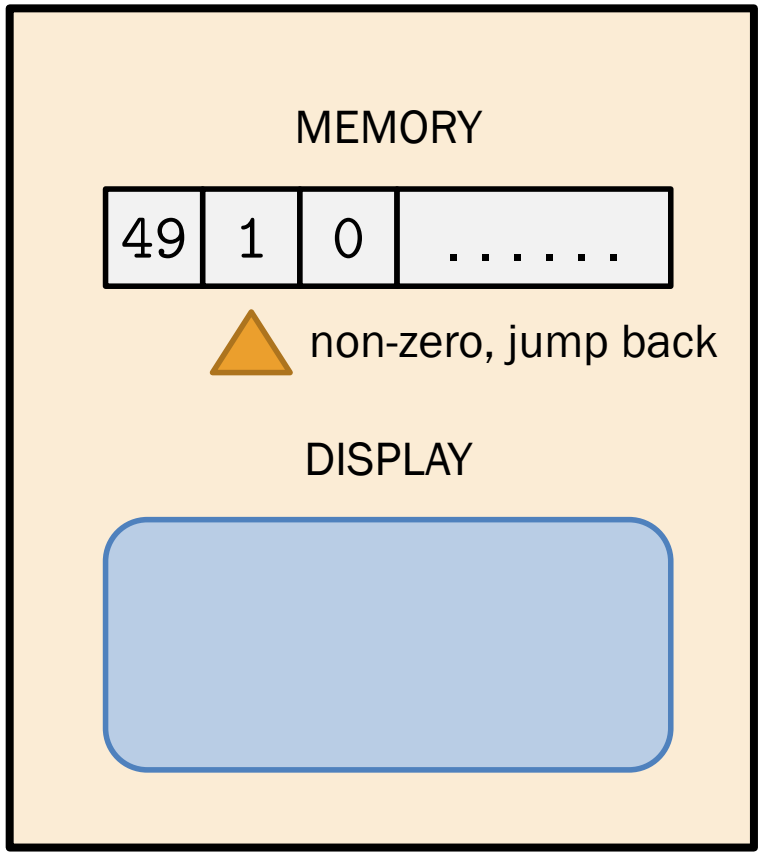


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

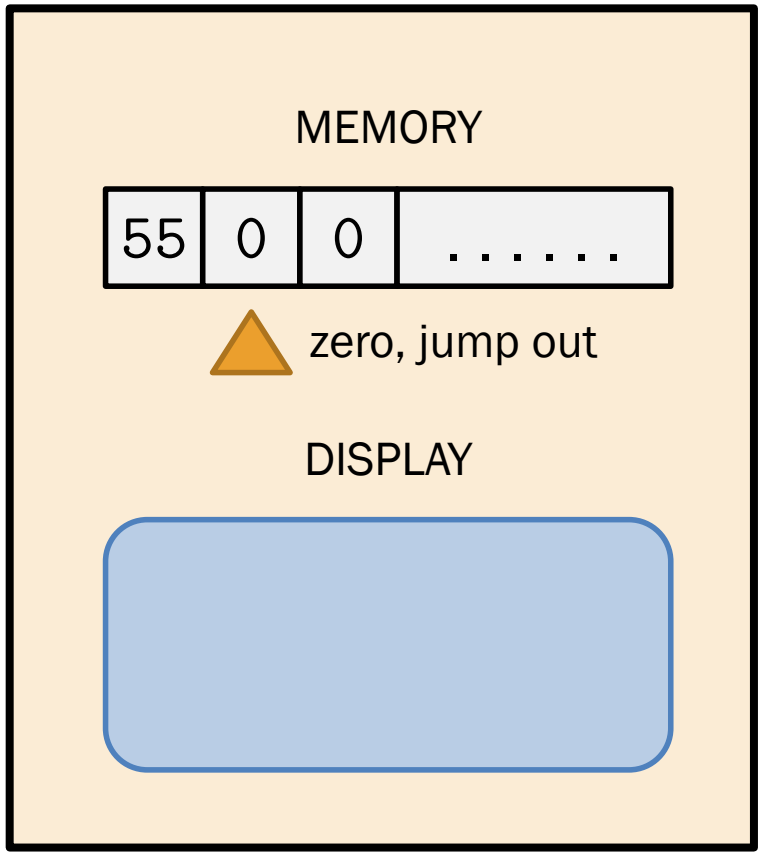


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

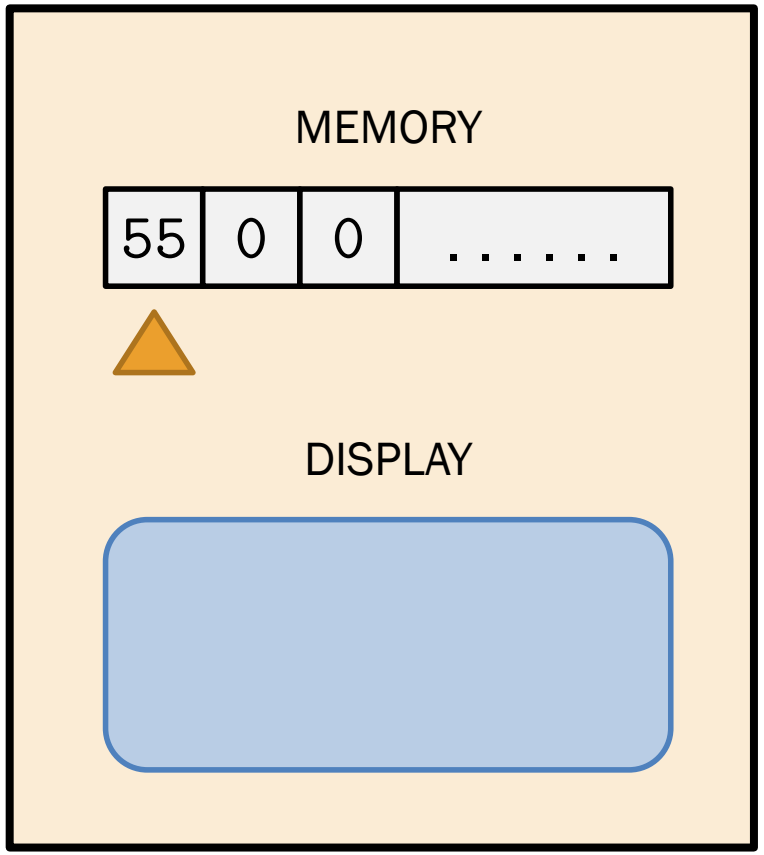


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```

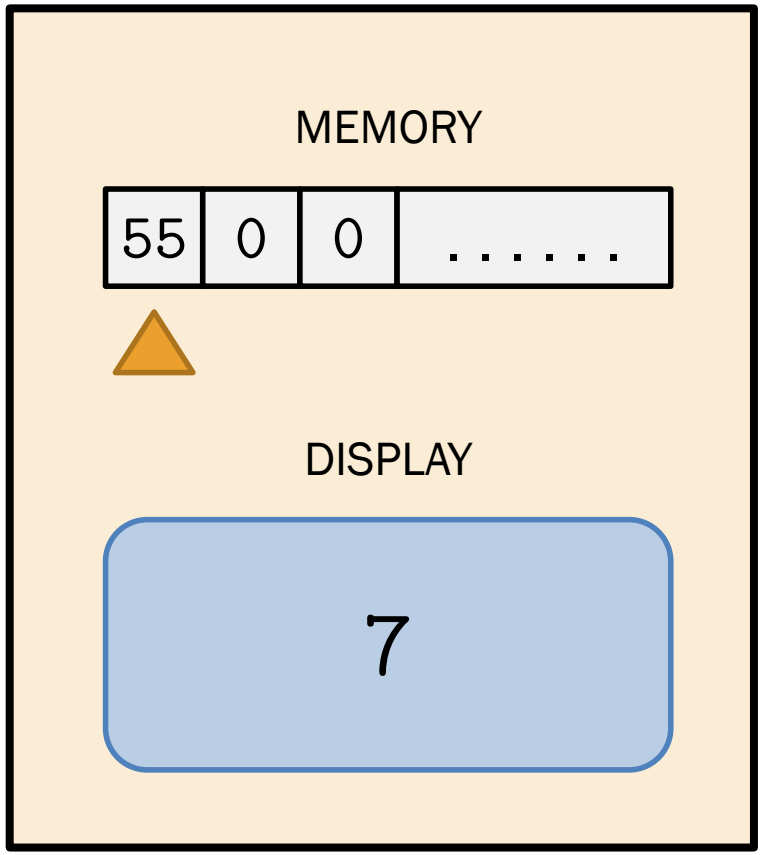


- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

```
++>+++++[<+>-]++++++++[<++++++>-]<.
```

How Does It Work?

```
++ >+++++ [<+>-]

++++++++

[

    < ++++++

    > -

]

<

.
```



- Three steps
  1. Compute 2+5
  2. Convert the result to ASCII
  3. Print it out

Anecdotes

- Brainfuck /ˈbreɪnfək ˈเบรนฟัค/
  - (*n.*) thing that is so complicated and unusual that it exceeds the limits of one's understanding
- Brainfuck has many derivatives e.g. JSFuck, BodyFuck, VerboseFuck, OooWee, Ook!, and Whitespace
- Brainfuck is Turing-complete

2.3 Jocular Languages

LOLCODE

- Created by Adam Lindsay in 2007
- Inspired by Lolcat internet meme, where the meme's text is deliberately grammatically incorrect

Wanna give it a shot?
Go to http://www.lolcode.org

```
HAI 1.1
  CAN HAS STDIO?
  VISIBLE "HAI WORLD!"
KTHXBYE
```

LOLCODE

```
#include <stdio>

int main() {
    std::cout << "HAI WORLD!";
}
```

Equivalent C++ code

Its syntax is designed for LOL

```
HAI 1.2
  CAN HAS STDIO?
  I HAS A NAME
  I HAS A HAIR_COLOR
  VISIBLE "WHAT IS YOUR NAME?"          BTW Ask for name
  GIMMEH NAME
  VISIBLE "WHAT IS YOUR HAIR COLOR?"    BTW Ask for hair color
  GIMMEH HAIR_COLOR
  HAIR_COLOR, WTF?                      BTW switch-case
    OMG "R"                             BTW case "R"
      VISIBLE NAME ", YOUR HAIR IS RED"
      GTFO                              BTW continue
    OMG "B"
      VISIBLE NAME ", YOUR HAIR IS BLACK"
      GTFO
    OMGWTF
      VISIBLE NAME ", YOUR HAIR IS WEIRD"
  OIC
KTHXBYE
```

Its syntax is designed for LOL

```
HOW IZ I FACTORIAL YR N
  DIFFRINT N AN SMALLR OF N AN 0       BTW If N > 0
  O RLY?
    YR RLY                             BTW True
      BTW Return N * FACTORIAL(N - 1)
      FOUND YR PRODUKT OF N AN FACTORIAL(DIFF OF N AN 1)
    NO WAI                             BTW False
      FOUND YR 1                       BTW Return 1
  OIC
IF U SAY SO

HAI 1.3
  CAN HAS STDIO?
  I HAS A N
  VISIBLE "ENTER N:"
  GIMMEH N
KTHXBYE
```

2.4 Poetic Languages

Shakespeare

- Created by Jon Åslund and Karl Wiberg in 2001
- Designed to resemble Shakespeare's plays, while equivalent to a verbose version of Assembly language

```
Outputting Input Reversedly.

Othello, a stacky man.
Lady Macbeth, who pushes him around till he pops.


                    Act I: The one and only.

                    Scene I: In the beginning, there was nothing.

[Enter Othello and Lady Macbeth]

Othello:
 You are nothing!

                    Scene II: Pushing to the very end.

Lady Macbeth:
 Open your mind! Remember yourself.

Othello:
 You are as hard as the sum of yourself and a stone wall. Am I as
 horrid as a flirt-gill?

Lady Macbeth:
 If not, let us return to scene II. Recall your imminent death!

Othello:
 You are as small as the difference between yourself and a hair!

                    Scene III: Once you pop, you can't stop!

Lady Macbeth:
 Recall your unhappy childhood. Speak your mind!

Othello:
 You are as vile as the sum of yourself and a toad! Are you better
 than nothing?

Lady Macbeth:
 If so, let us return to scene III.

                    Scene IV: The end.

[Exeunt]
```

Playwright as Coding

```
Do Not Adieu, a play in two acts.
```
Title (program's name)

```
Romeo, a young man with a remarkable patience.
Juliet, a likewise young woman of remarkable grace.
Ophelia, a remarkable woman much in dispute with Hamlet.
Hamlet, the flatterer of Andersen Insulting A/S.
```
Variable declaration

```
Act I: Hamlet's insults and flattery.

Scene I: The insulting of Romeo.
```
Label for GOTO statement

```
[Enter Hamlet and Romeo]
```
Call variables to the stage

```
Hamlet:
 You lying stupid fatherless big smelly half-witted coward!
 You are as stupid as the difference between a handsome rich brave
 hero and thyself! Speak your mind!

 You are as brave as the sum of your fat little stuffed misused dusty
 old rotten codpiece and a beautiful fair warm peaceful sunny summer's
 day. You are as healthy as the difference between the sum of the
 sweetest reddest rose and my father and yourself! Speak your mind!

 You are as cowardly as the sum of yourself and the difference
 between a big mighty proud kingdom and a horse. Speak your mind.

 Speak your mind!
```
Variable assignment & Output

```
[Exit Romeo]
```
Let variable leave the scene

- Variable assignment
  - Constants are represented by combinations of nouns and adjectives
  - Positive and neutral nouns = 1 ; negative nouns = -1
  - Each adjective multiplies a noun by 2
  - Pronouns 'you' & 'thou' = variable
- I/O
  - Speak your mind = print number
  - Open your heart = print string
  - Listen to your heart = input number
  - Open your mind = input string
- GOTO
  - Let us / We shall / We must + return to / proceed to + Act/Scene

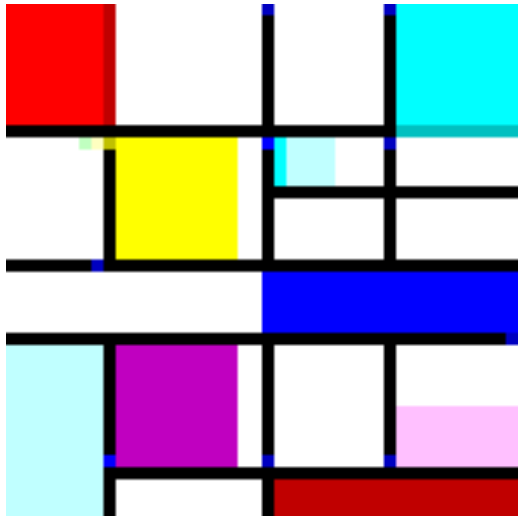
2.5 Pictorial Languages

Piet /pʰiːt/

- Code disguised as abstract painting
  - Instructions are encoded as the change of hue (color) and lightness
  - Program pointer starts from the upper-left corner, traverses from L to R by default, and turns 90° CW once it hits a border or a black cell
  - The program terminates when the pointer cannot move anywhere else

#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

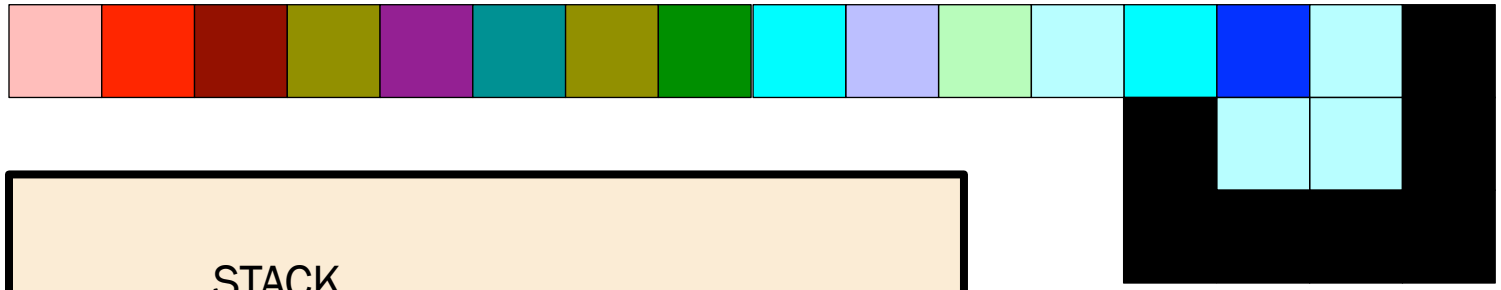
	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

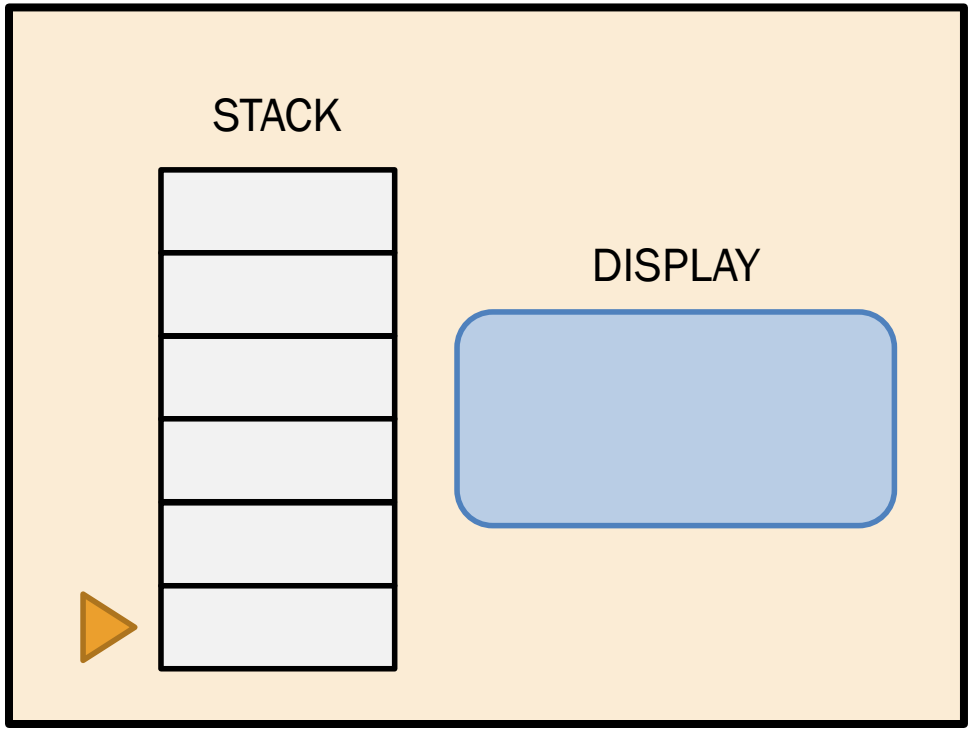


This code prints out “Piet”

Wanna give it a shot? Go to https://www.dangermouse.net/esoteric/piet.html

How Does It Work?





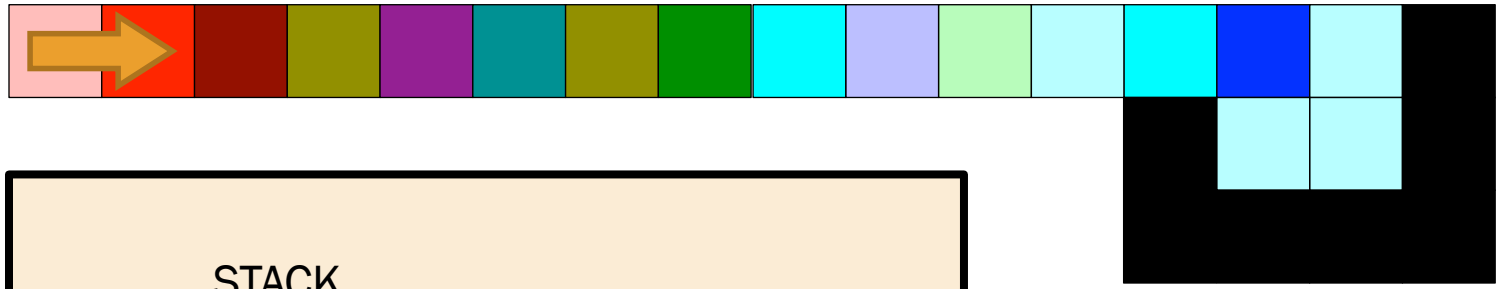
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

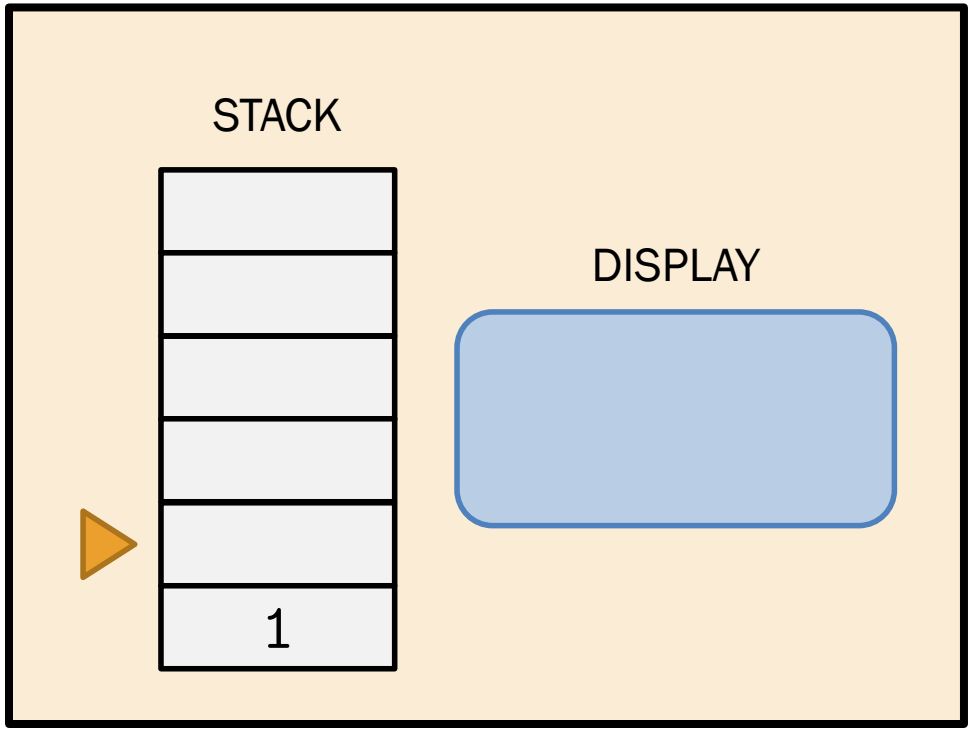
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 0, lightness 1: push 1





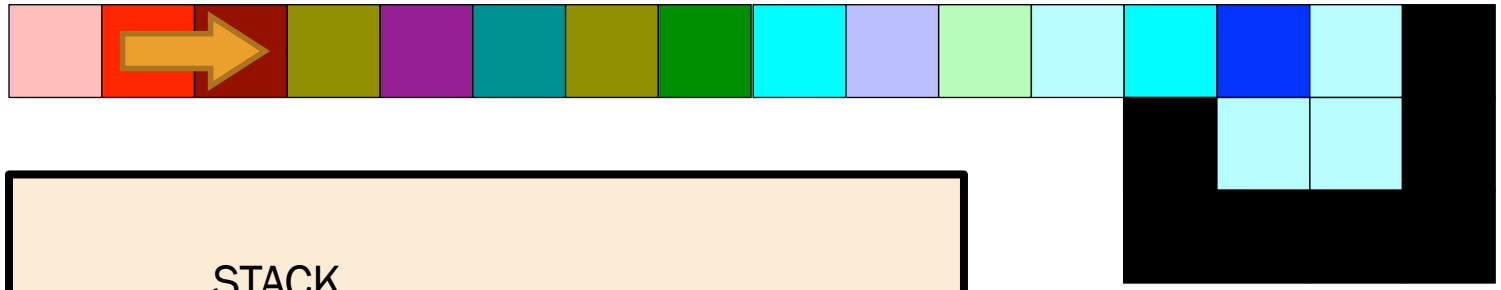
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

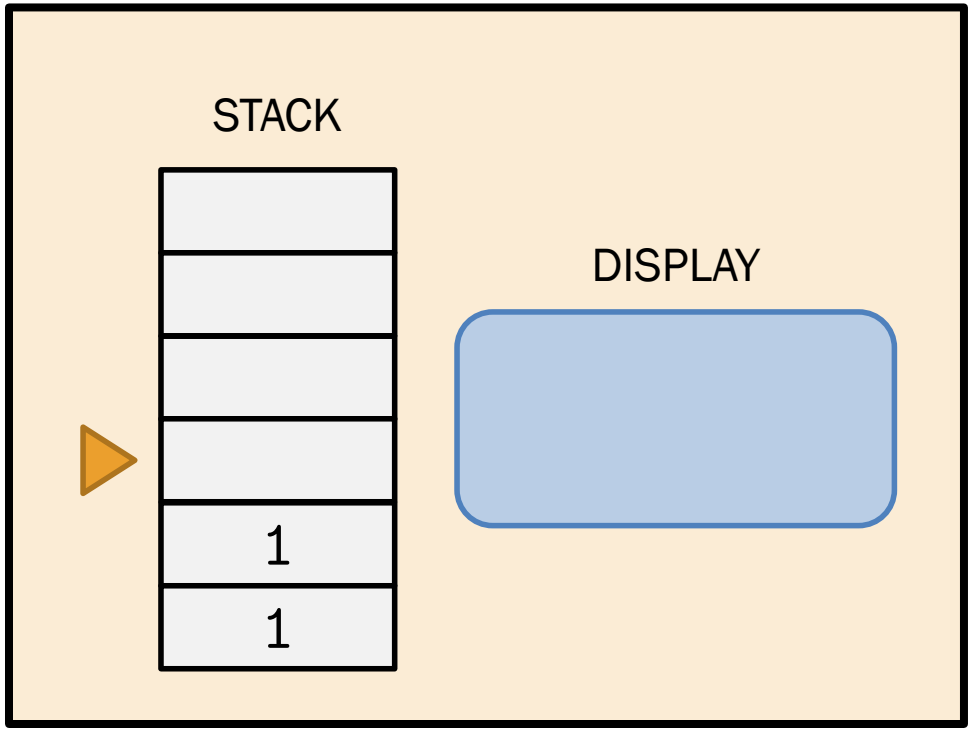
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 0, lightness 1: push 1





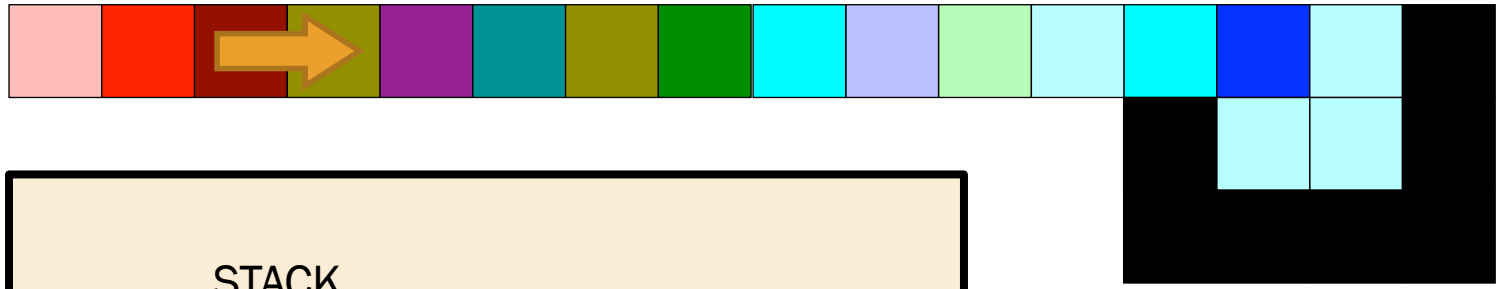
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

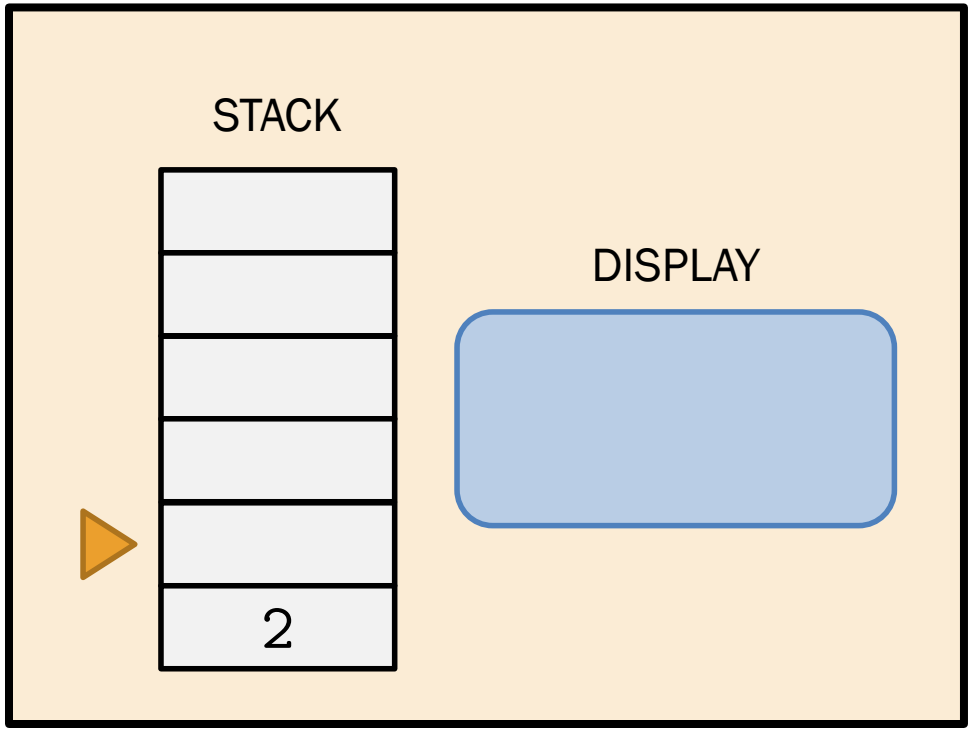
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 1, lightness 0: postfix add





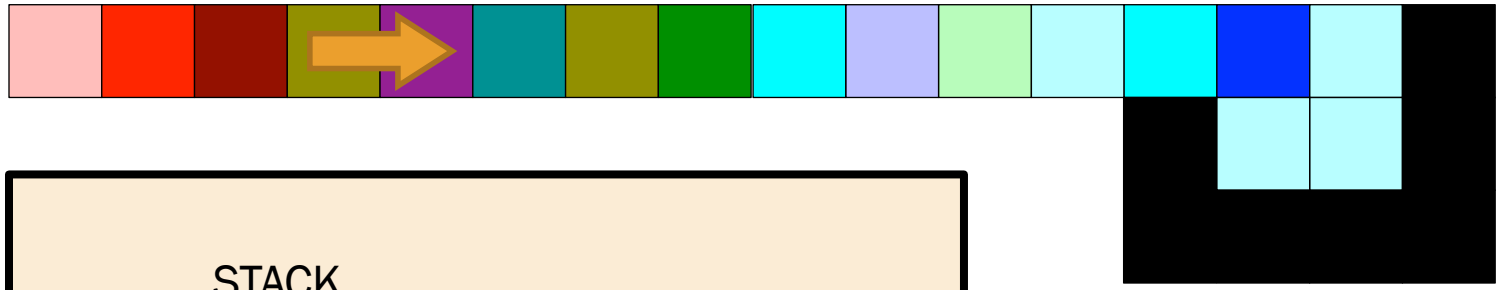
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

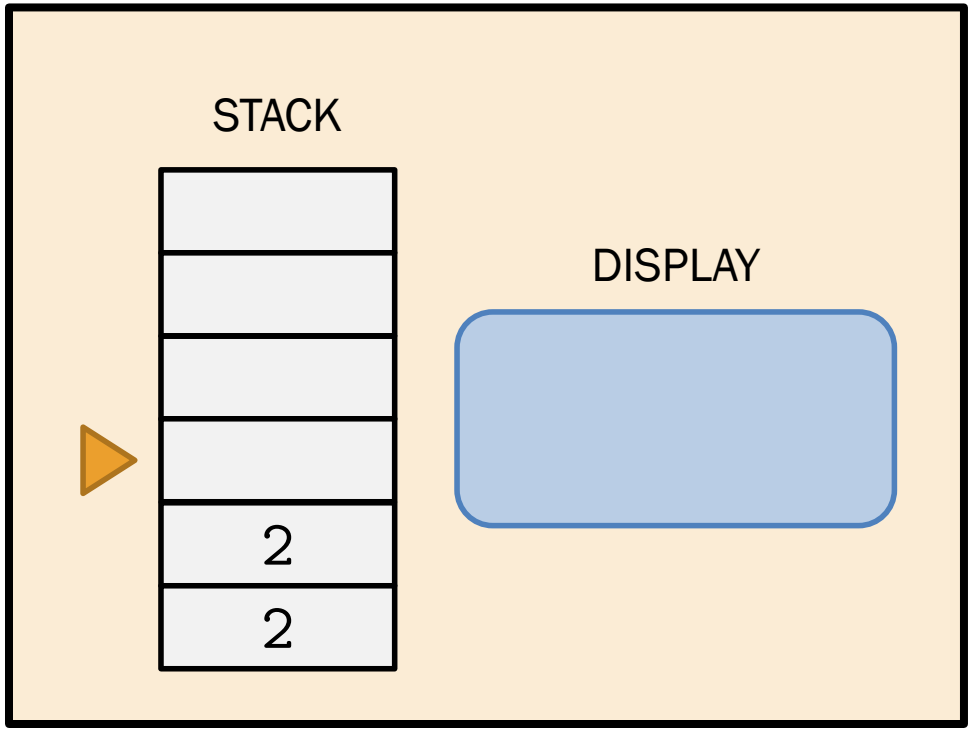
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 4, lightness 0: duplicate top





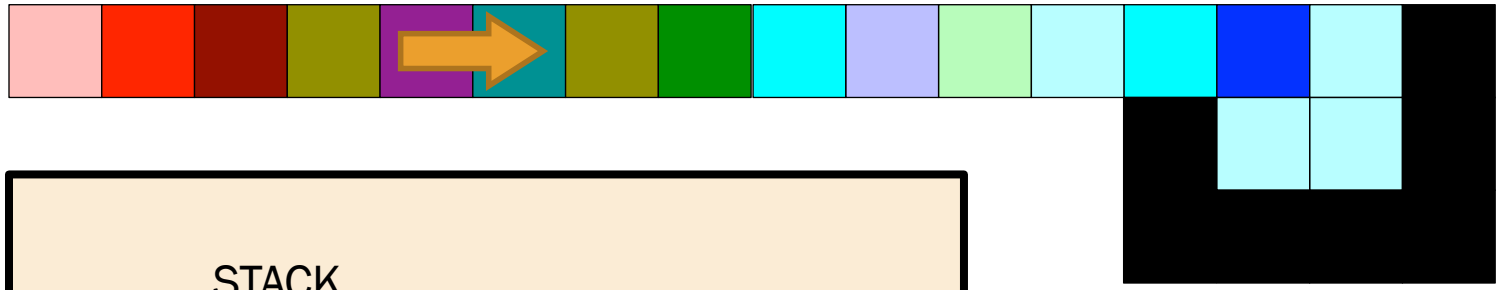
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

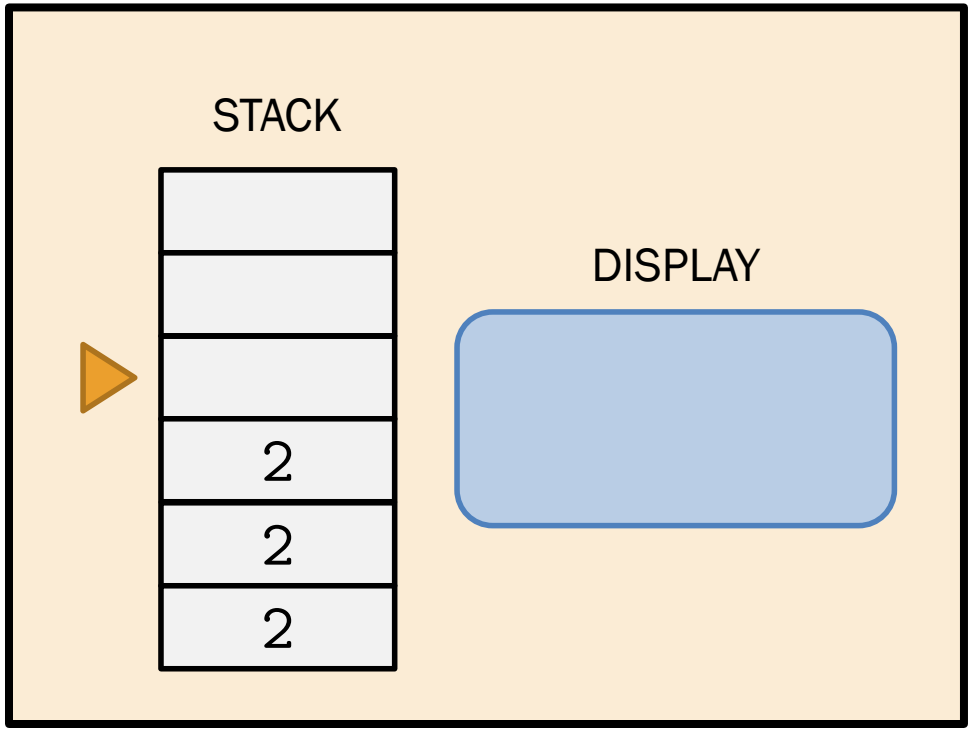
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 4, lightness 0: duplicate top





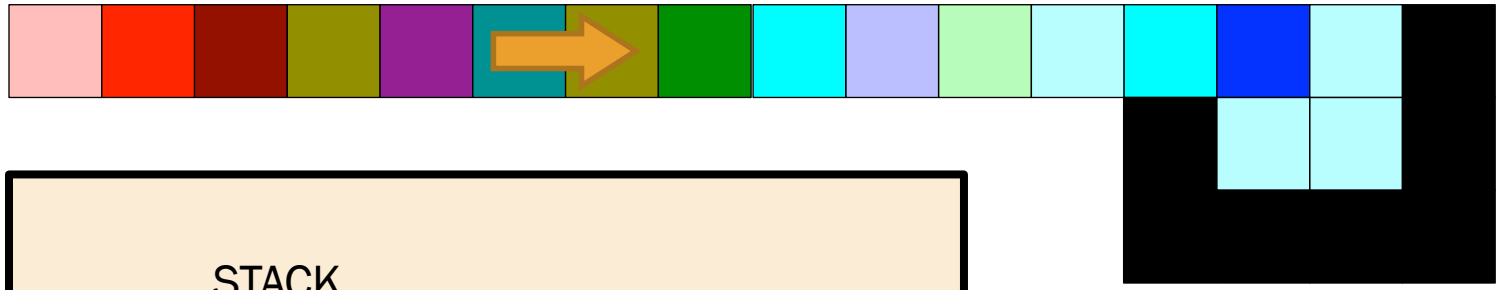
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

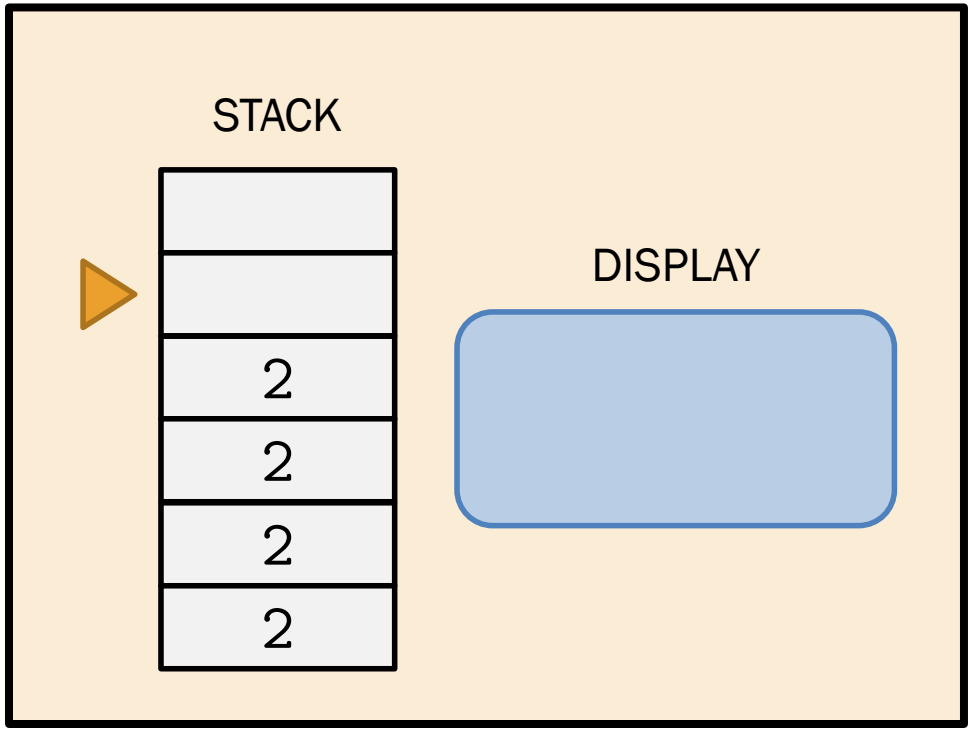
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 4, lightness 0: duplicate top





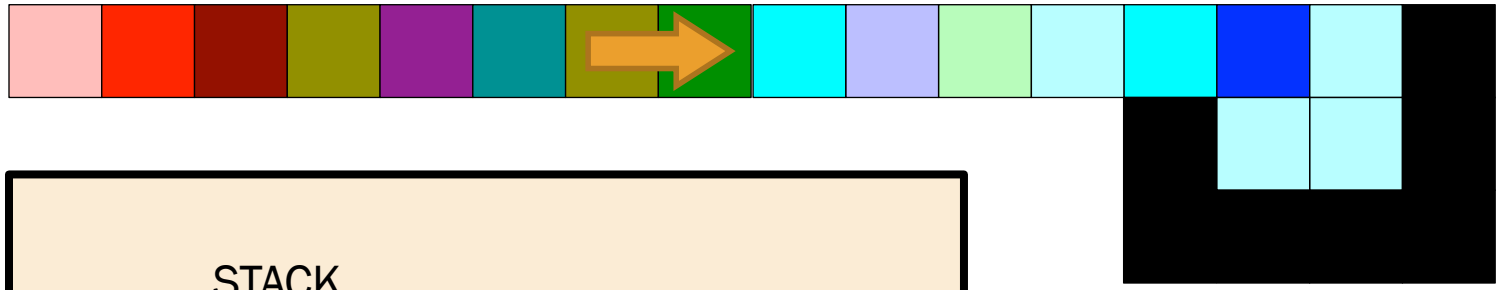
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

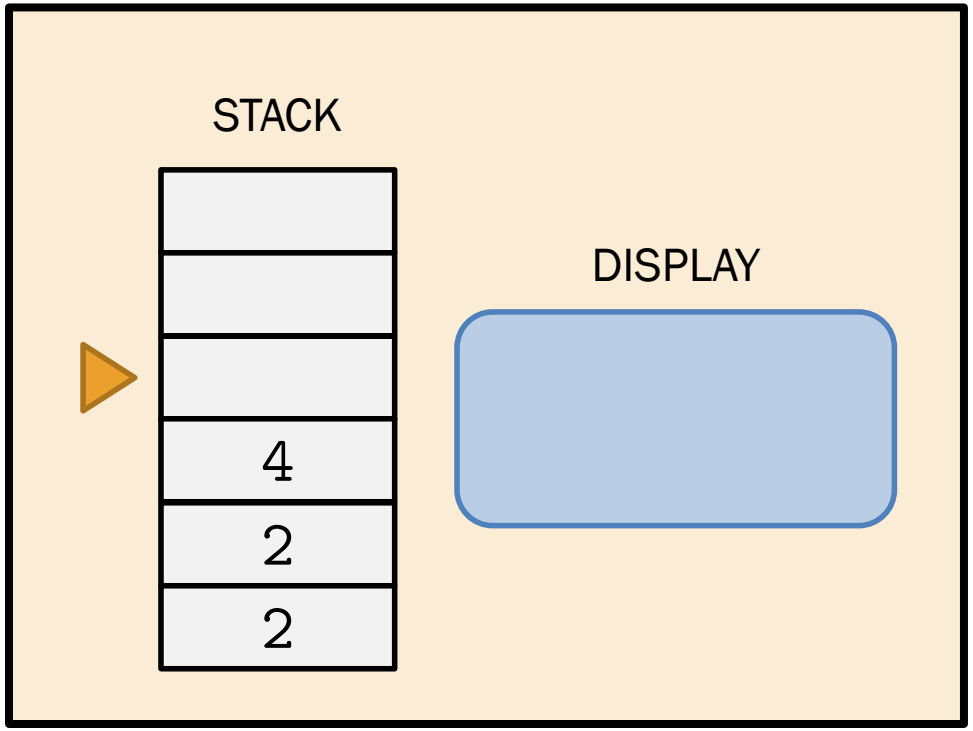
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 1, lightness 0: add





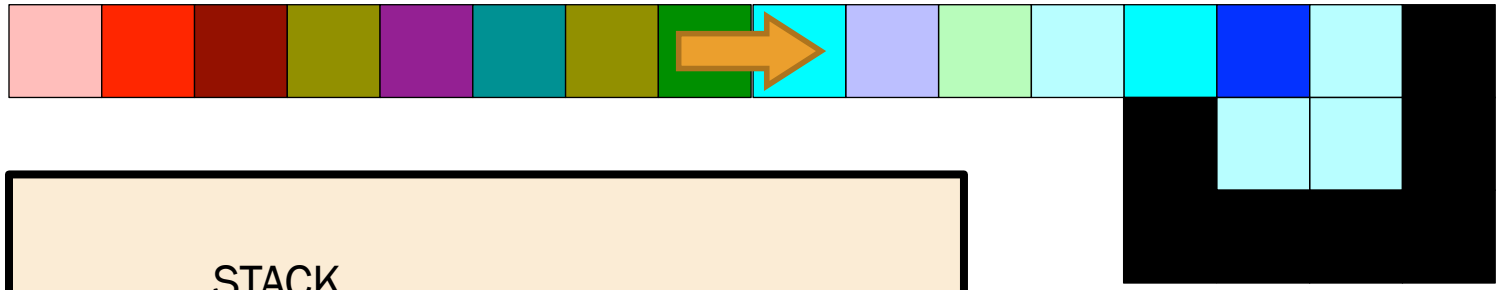
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

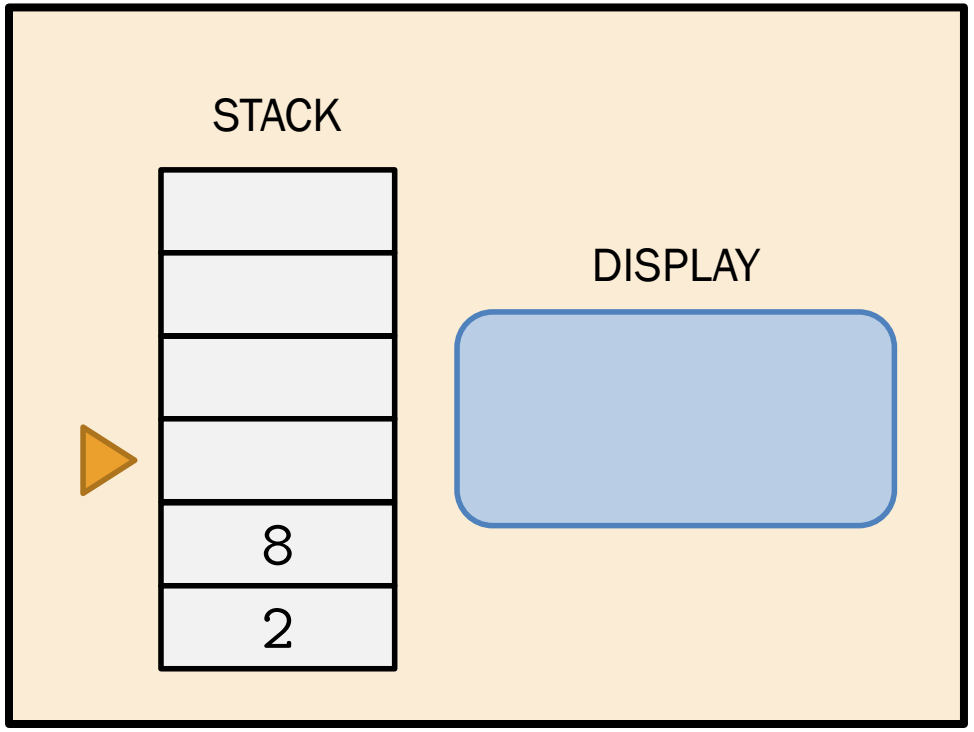
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 1, lightness 2: multiply





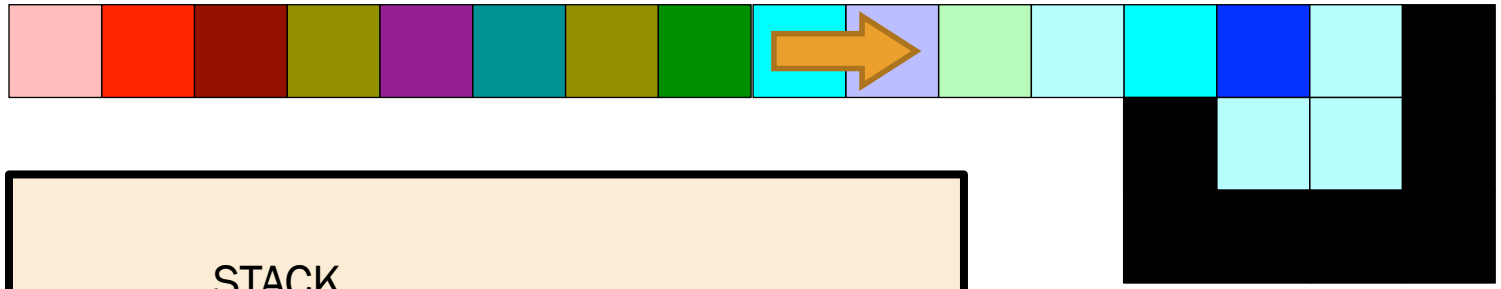
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

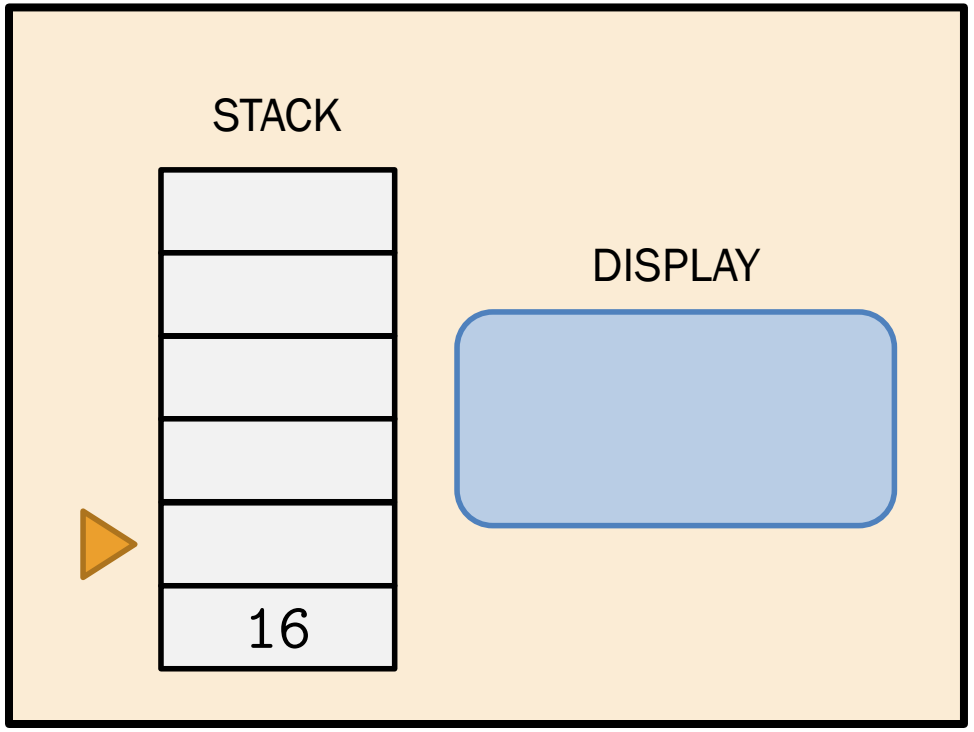
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 1, lightness 2: multiply





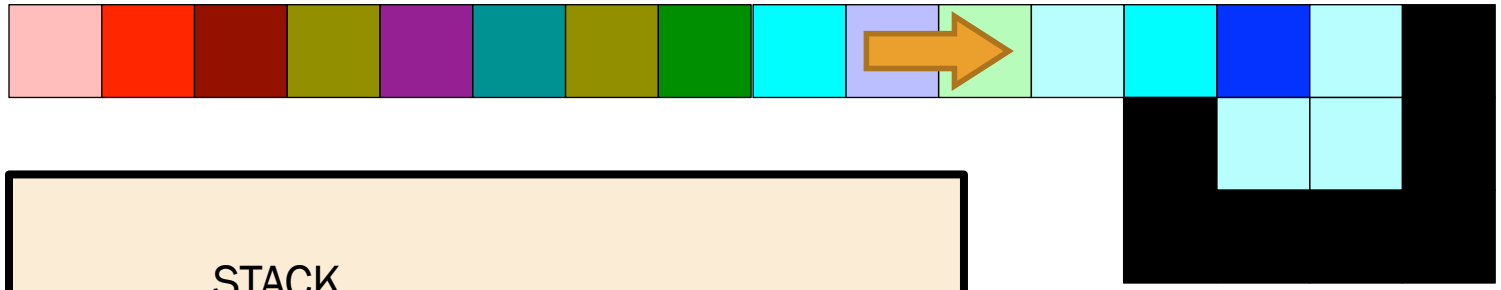
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

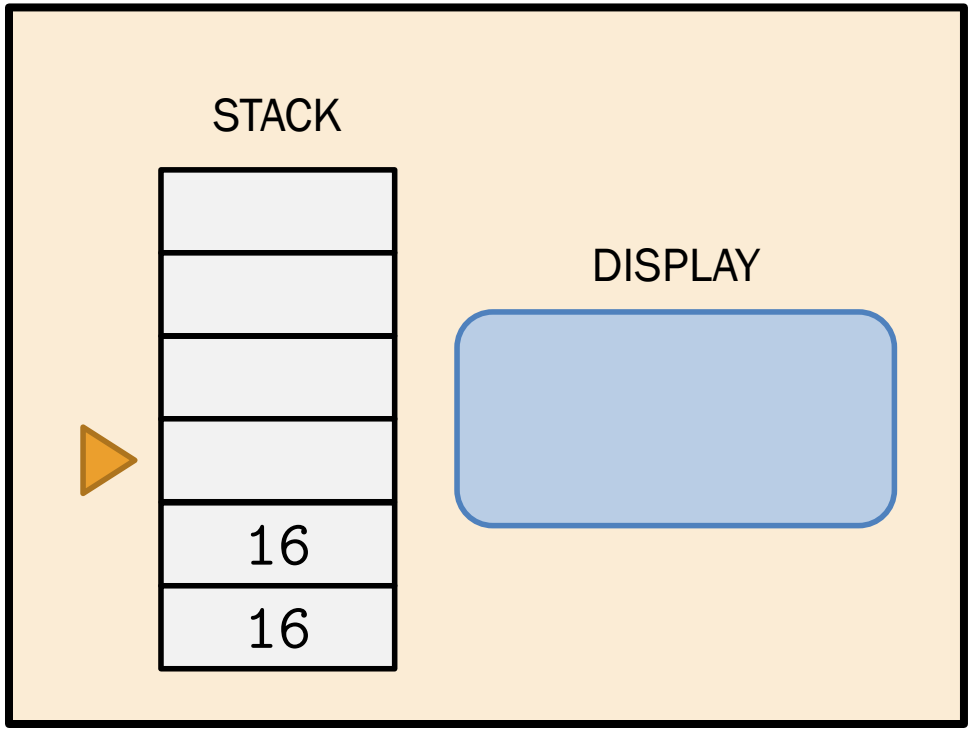
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 4, lightness 0: duplicate top





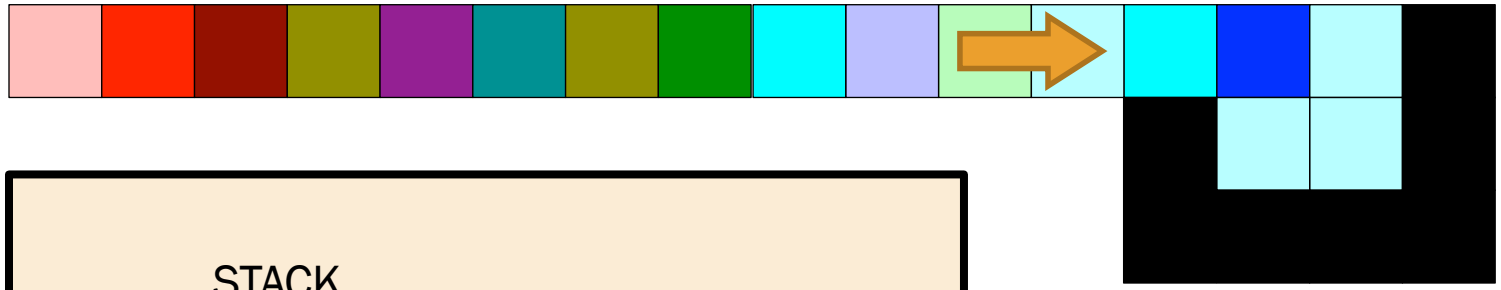
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

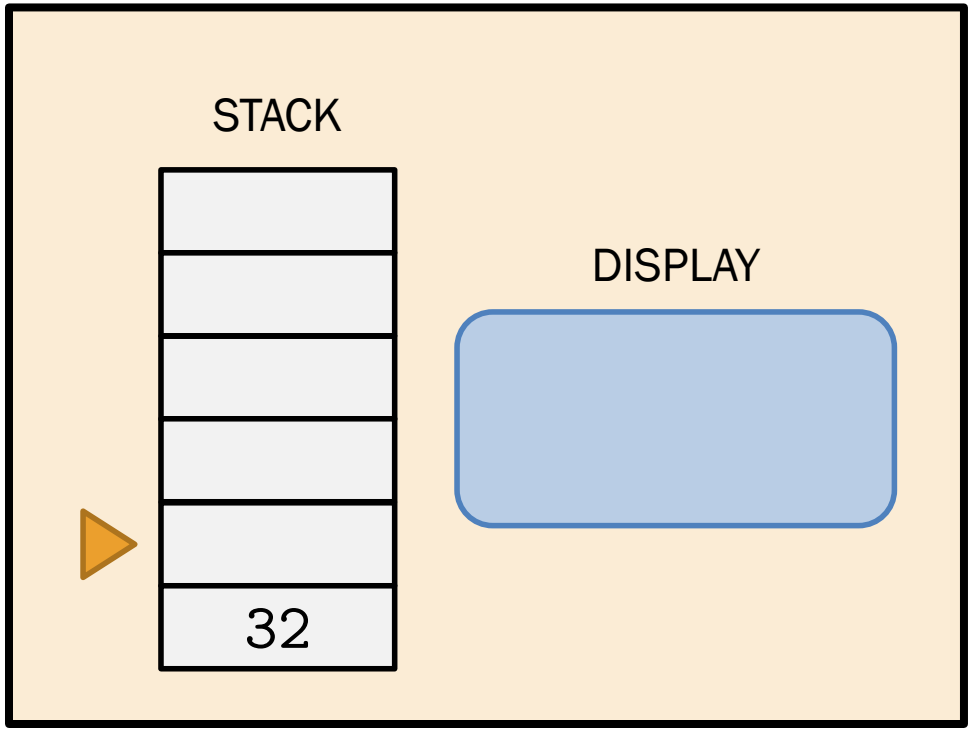
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 1, lightness 0: postfix add





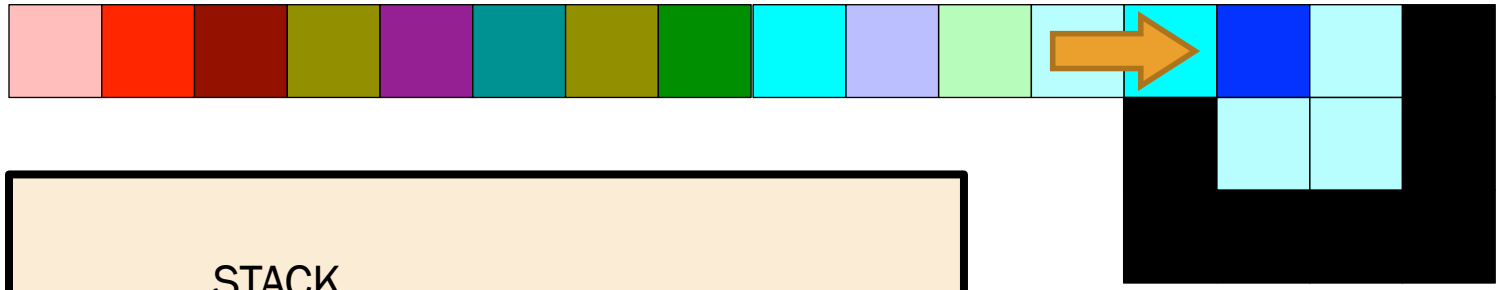
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

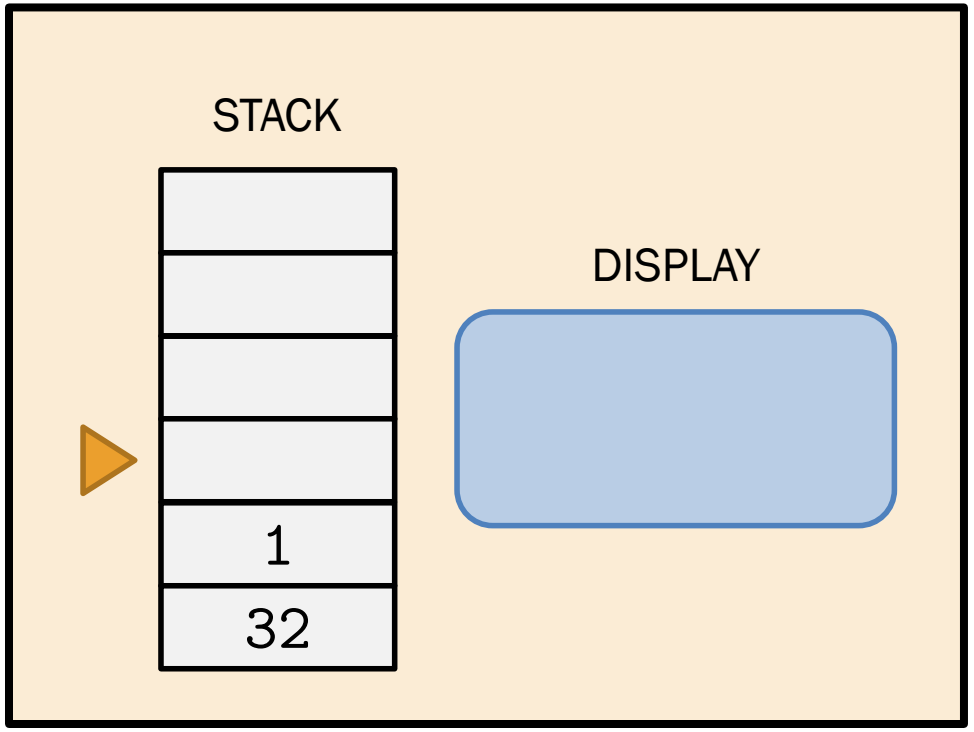
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 0, lightness 1: push 1





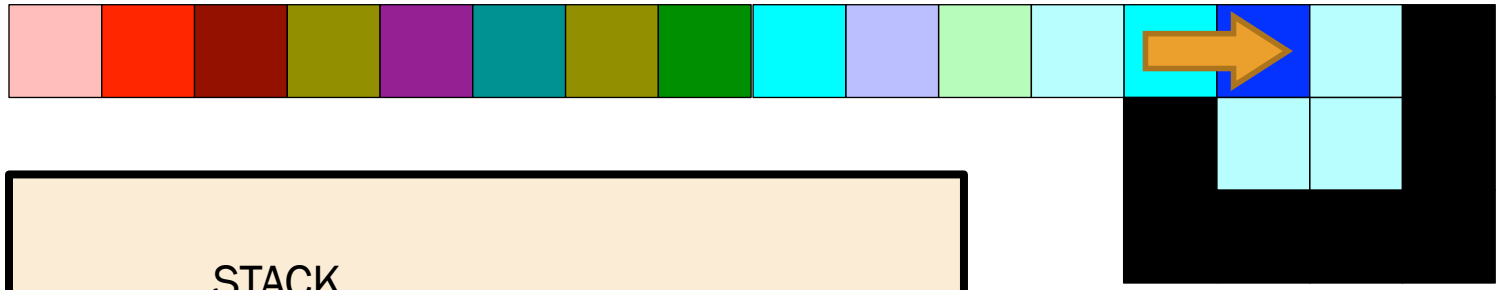
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

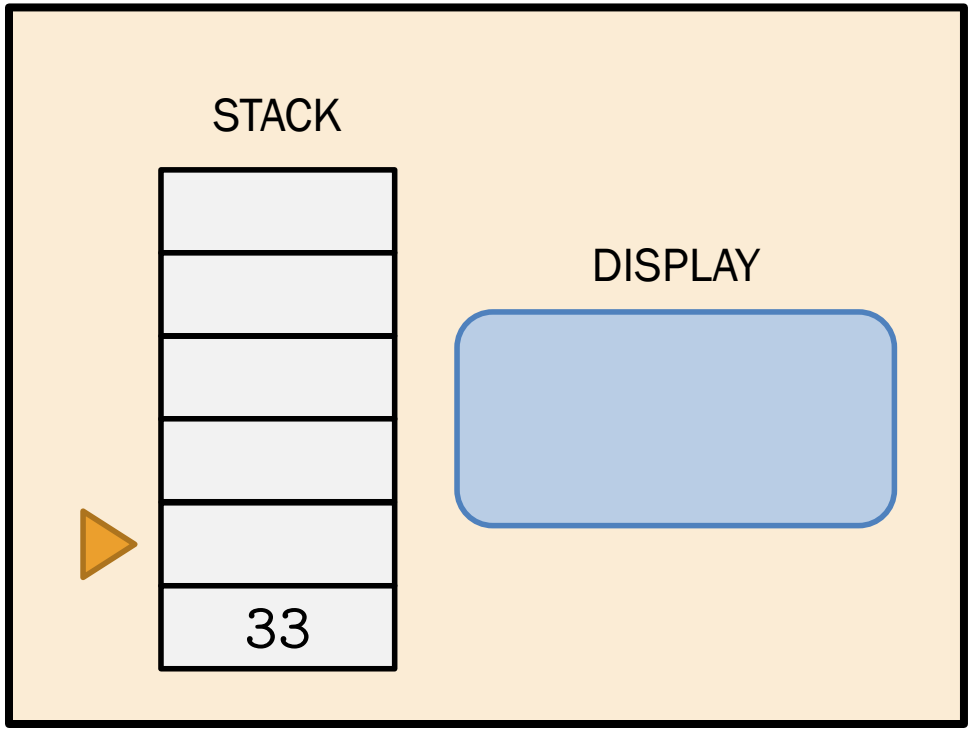
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 1, lightness 0: postfix add





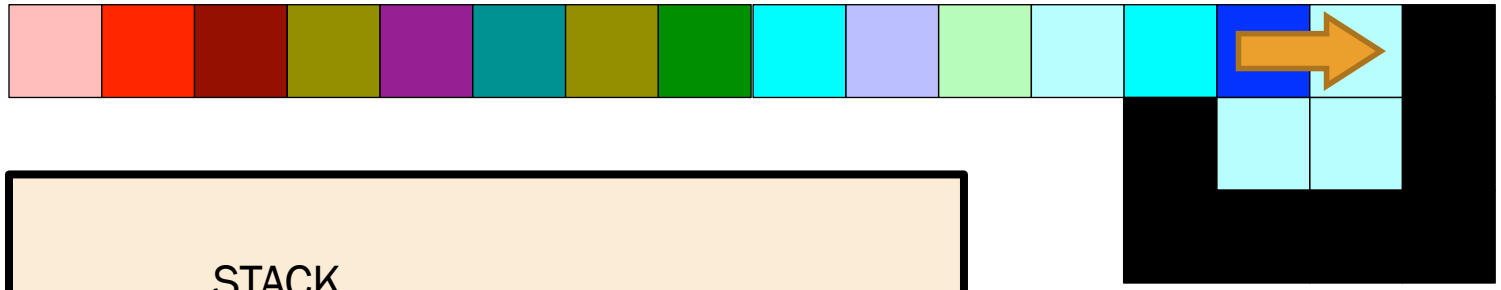
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

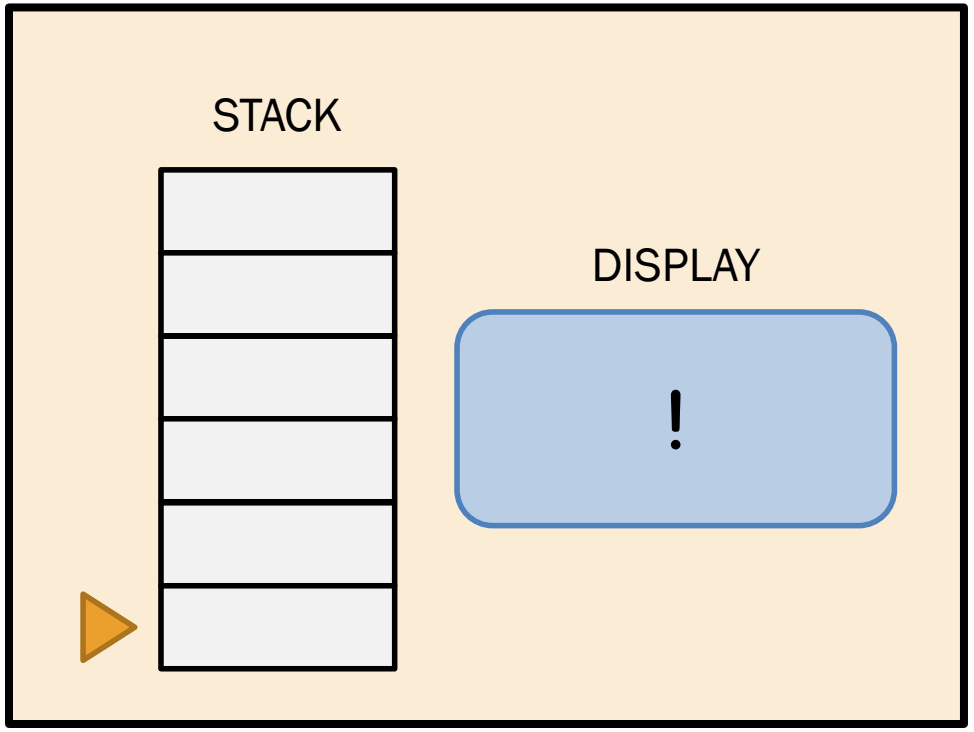
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 5, lightness 2: output ASCII





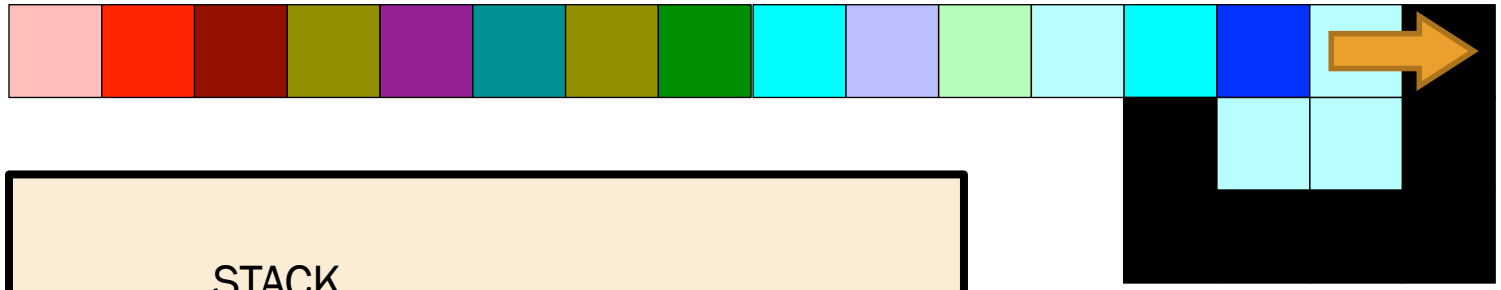
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

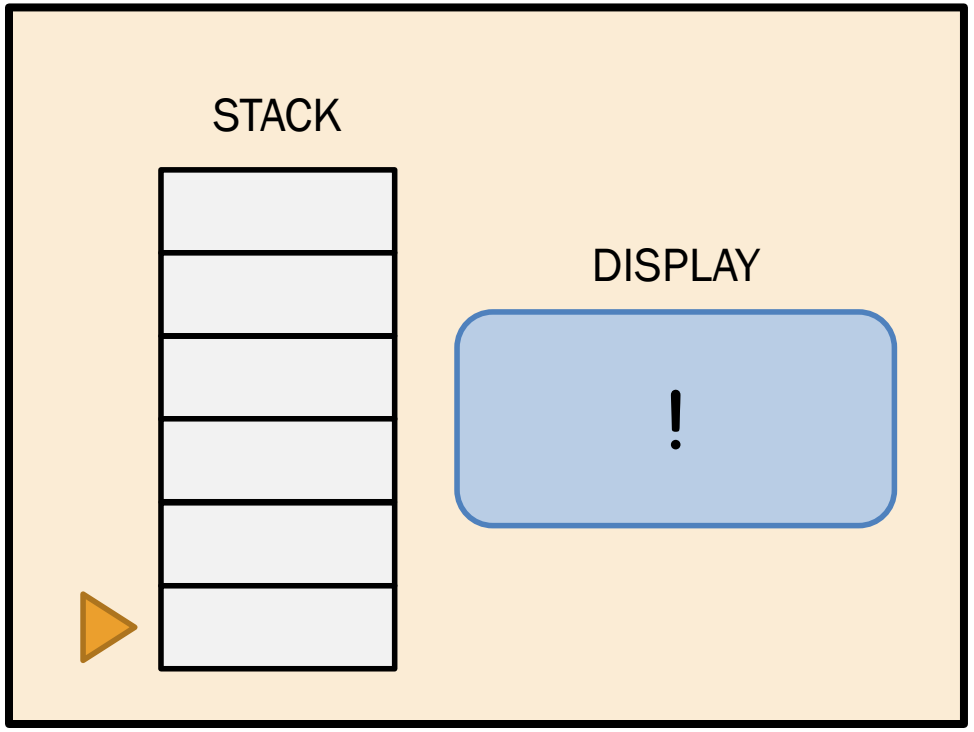
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

blocked by black cells: rotate 90°CW





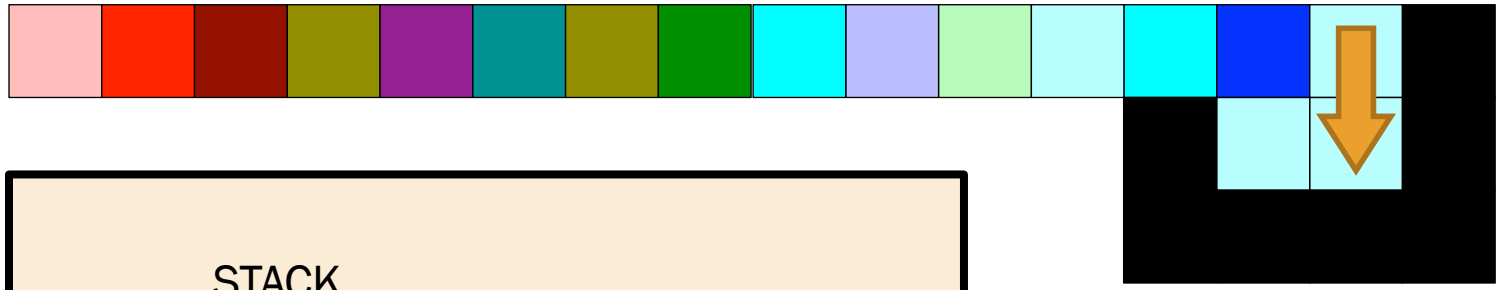
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

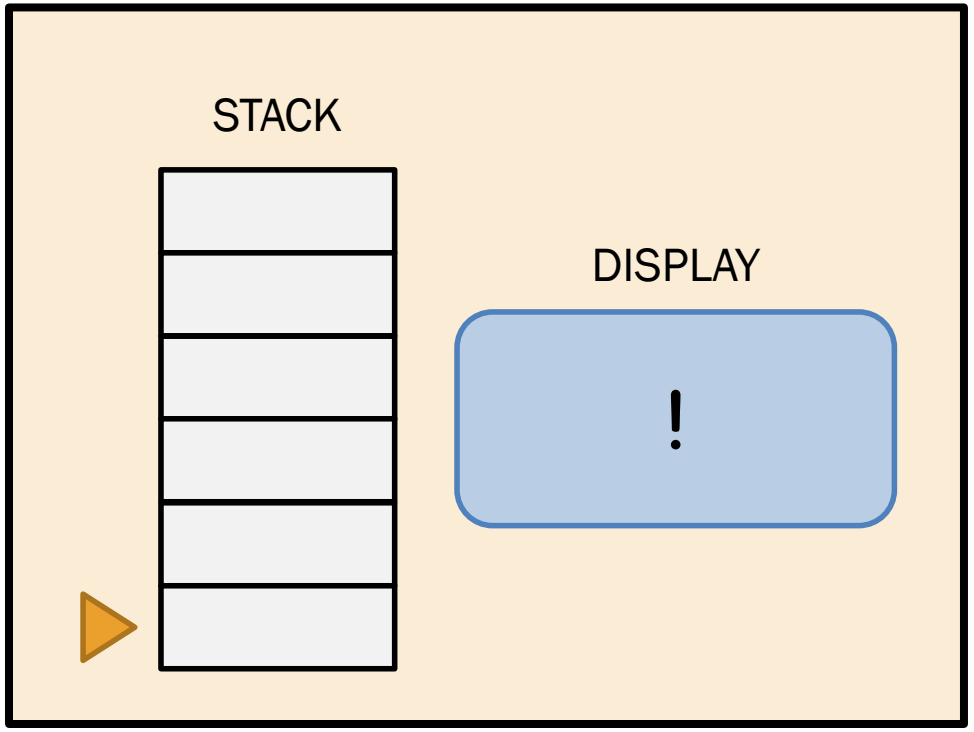
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

now we have rotated 90°CW





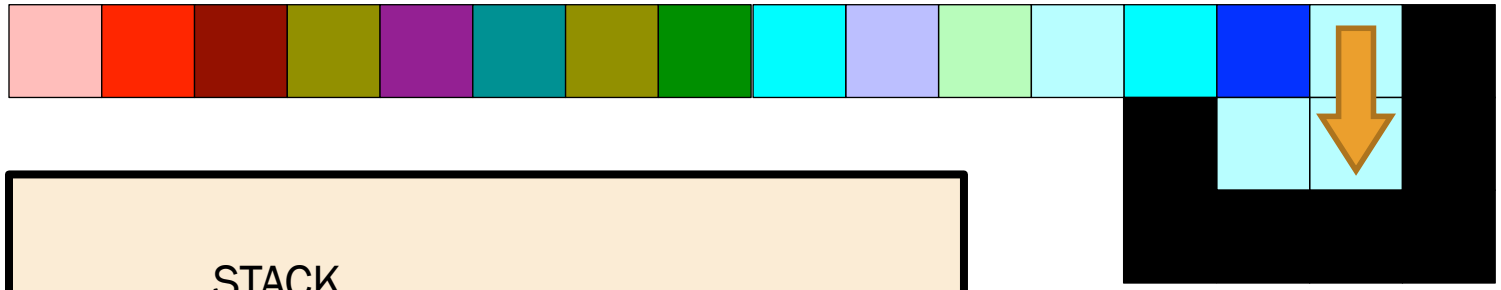
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

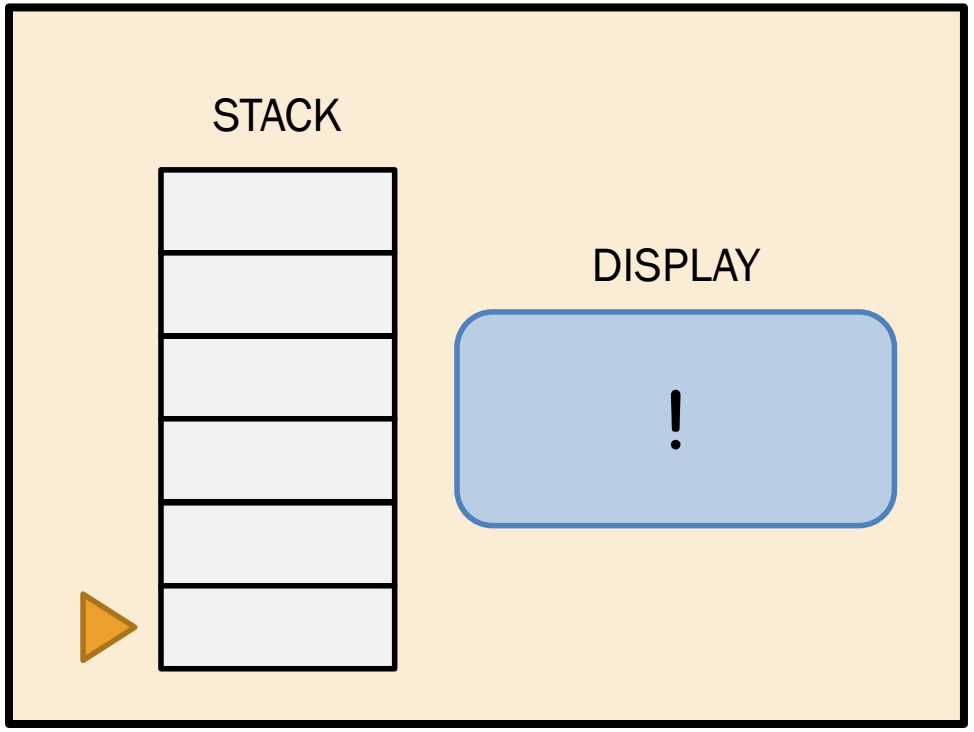
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 0, lightness 0: no operations





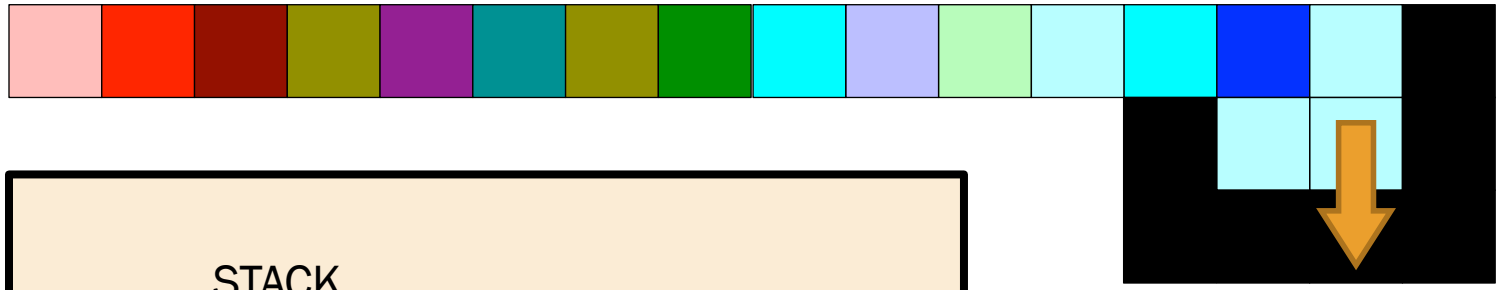
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

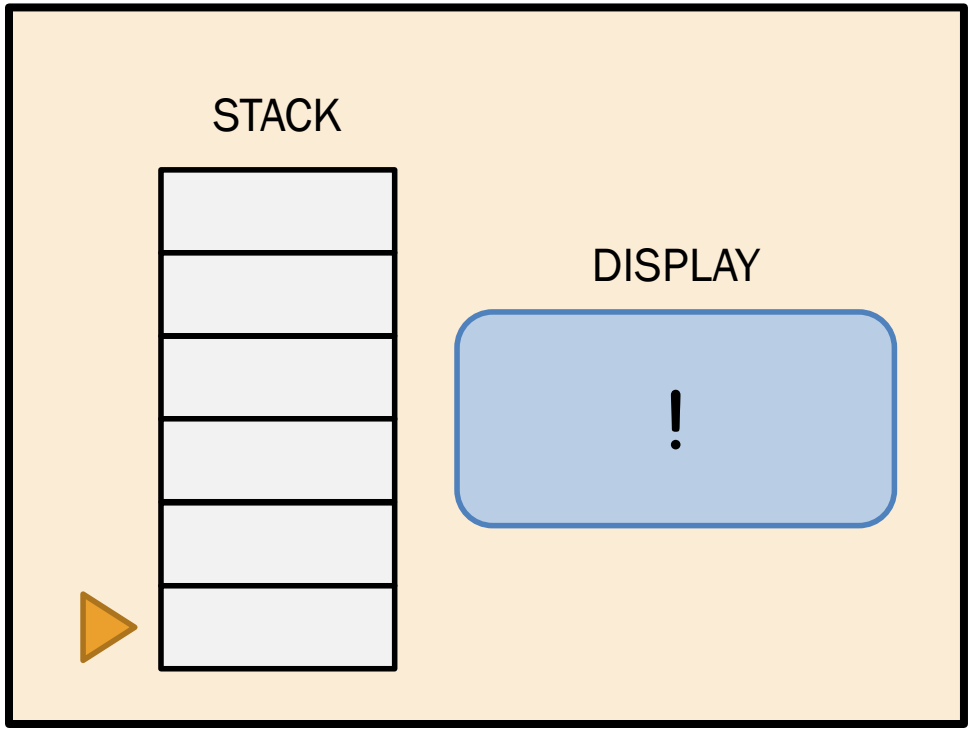
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

blocked by black cells: rotate 90°CW





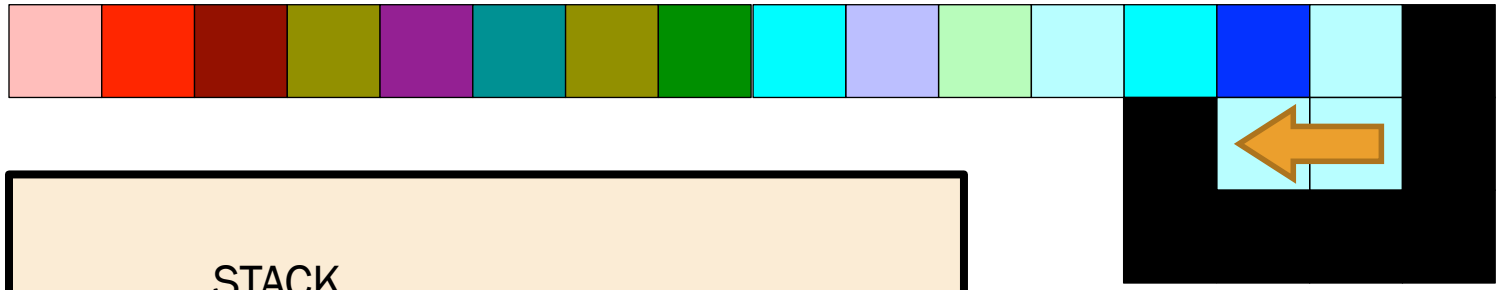
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

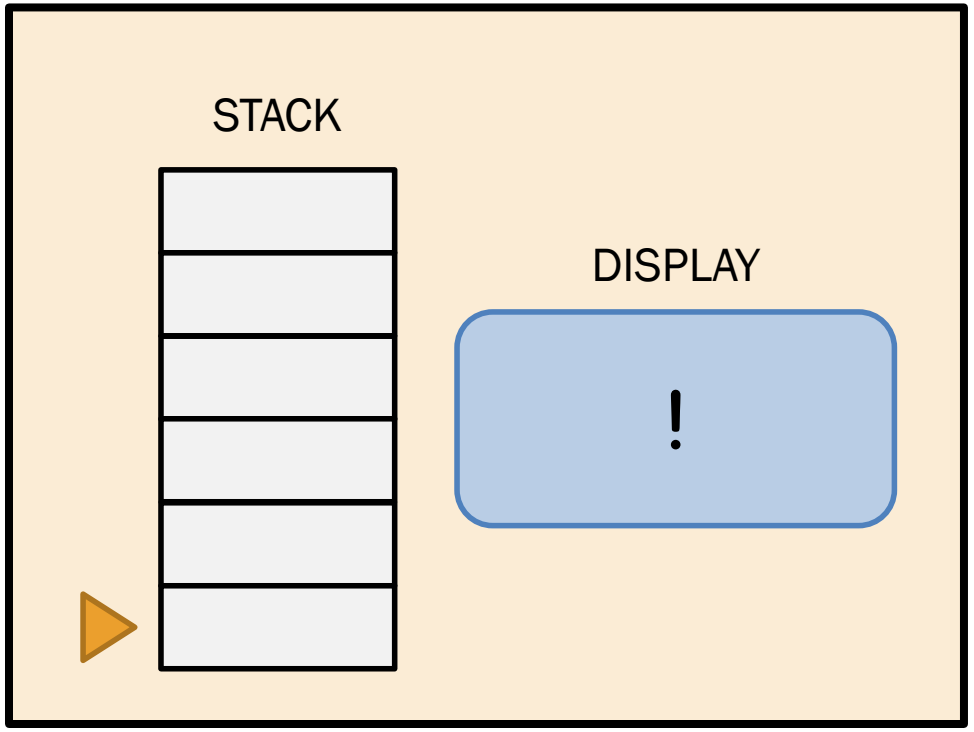
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

hue 0, lightness 0: no operations





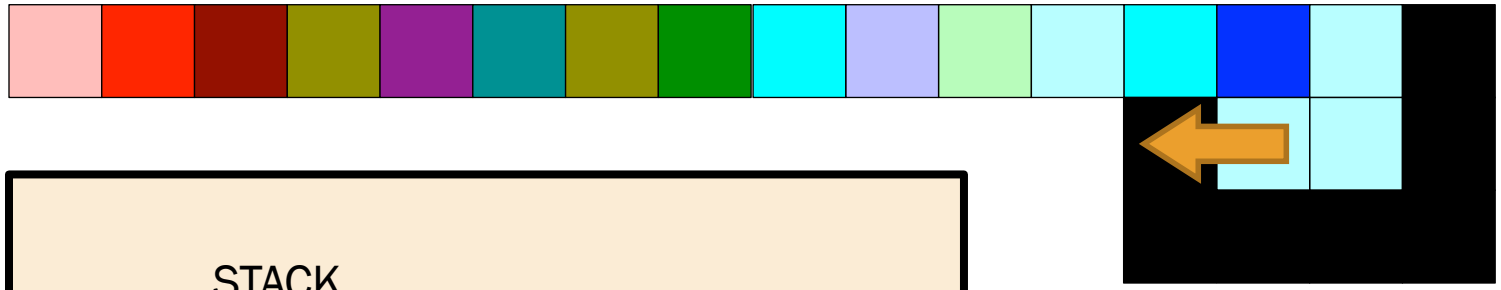
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

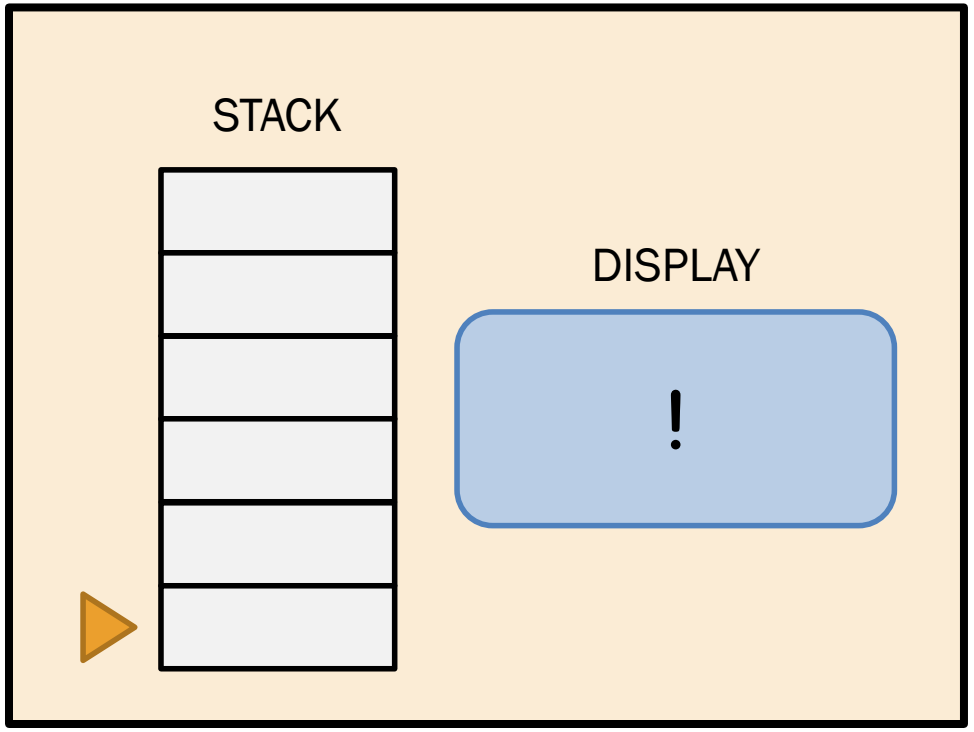
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

blocked by black cells: rotate 90°CW





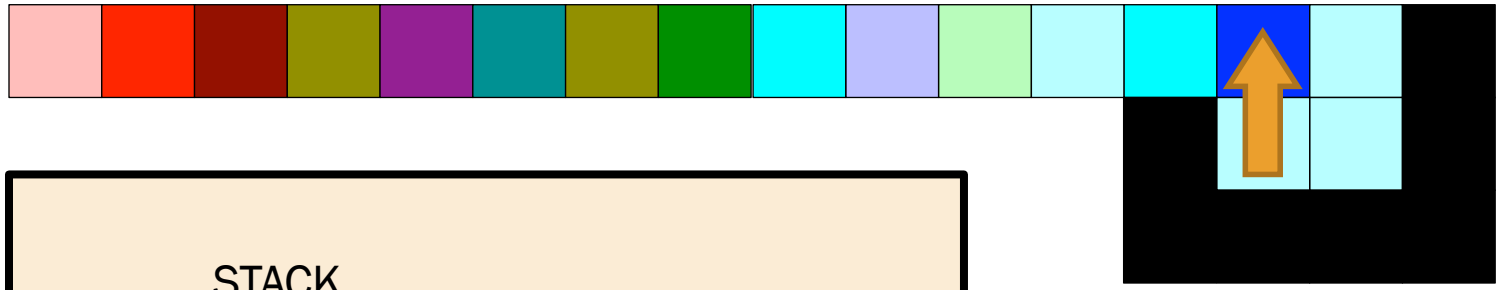
#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

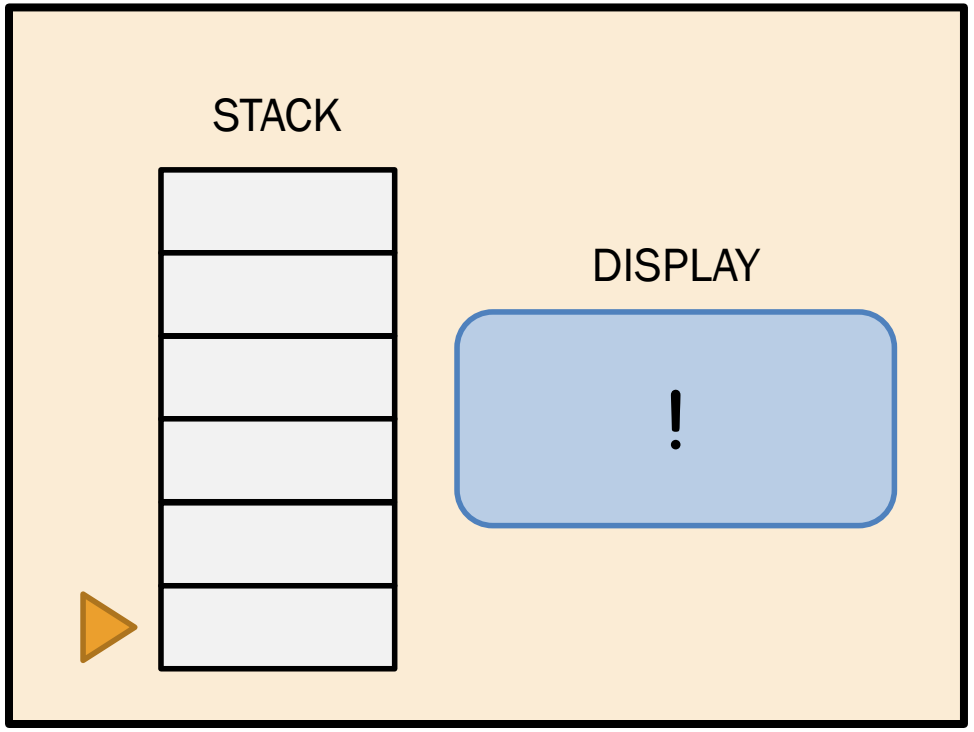
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

How Does It Work?

visited cell, and no more turns to make: terminate





#FFC0C0 light red	#FFFFC0 light yellow	#C0FFC0 light green	#C0FFFF light cyan	#C0C0FF light blue	#FFC0FF light magenta
#FF0000 red	#FFFF00 yellow	#00FF00 green	#00FFFF cyan	#0000FF blue	#FF00FF magenta
#C00000 dark red	#C0C000 dark yellow	#00C000 dark green	#00C0C0 dark cyan	#0000C0 dark blue	#C000C0 dark magenta
#FFFFFF white			#000000 black		

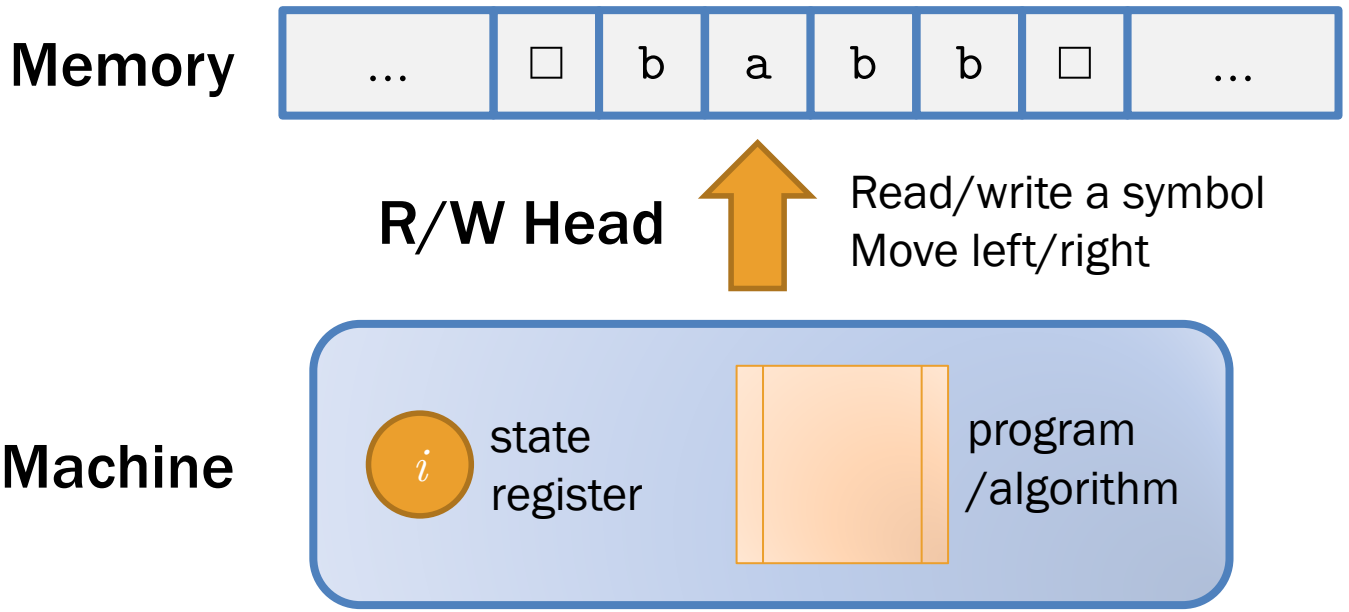
- **Hue Cycle:** red -> yellow -> green -> cyan -> blue -> magenta -> red
- **Lightness Cycle:** light -> normal -> dark -> light

	Lightness change		
Hue change	**None**	**1 Darker**	**2 Darker**
None		push	pop
1 Step	add	subtract	multiply
2 Steps	divide	mod	not
3 Steps	greater	pointer	switch
4 Steps	duplicate	roll	in(number)
5 Steps	in(char)	out(number)	out(char)

3. Discussion

Recursively Enumerable Languages

- Language whose each string is produced by a mathematical function that can be defined by recursion (i.e. an algorithm)
- Recognized by Turing machines



- We put an input string in the memory
- In each operation
  - R/W head manipulates the content of the current cell
  - R/W head moves left or right
- At the end, the machine will notify **accept** or **fail**

Recursively Enumerable Languages

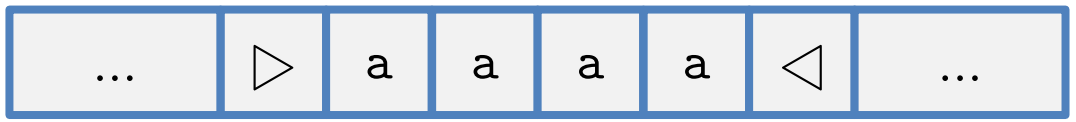
- Examples
  - Multiply language $L_{\text{mult}} = \{\mathtt{a}^p\mathtt{b}^q\mathtt{c}^r \mid r = pq\}$
  - Polynomial language $L_{\text{poly}} = \{\mathtt{a}^{n^k} \mid n \geq 0\}$
  - Exponential language $L_{\text{exp}} = \{\mathtt{a}^{2^n} \mid n \geq 0\}$
  - Prime language $L_{\text{prime}} = \{\mathtt{a}^p \mid p \text{ is prime}\}$
  - etc.
- In recognizing these languages, we have to count the number of a's and check it against a recursively defined function

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

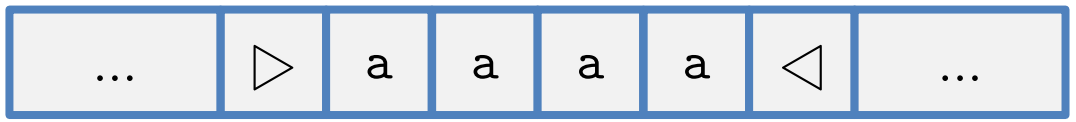
Memory



Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Memory



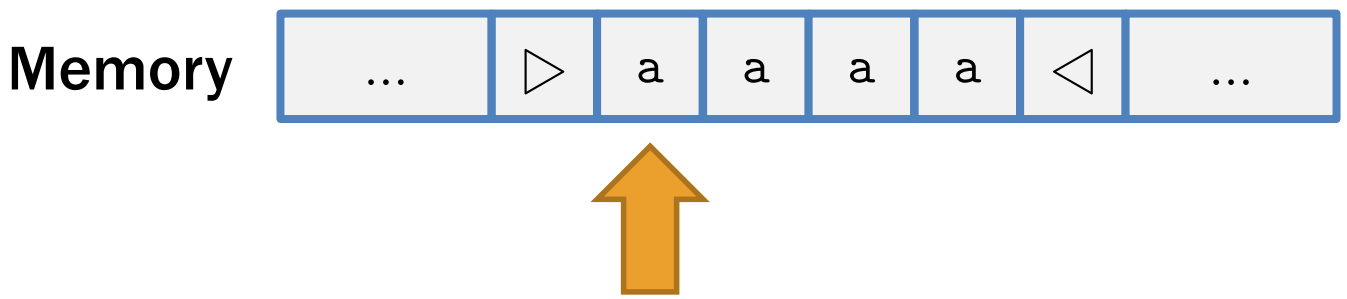
Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

The input string is put in the memory wrapped with ▷ and ◁ to mark the beginning and the end of string

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.



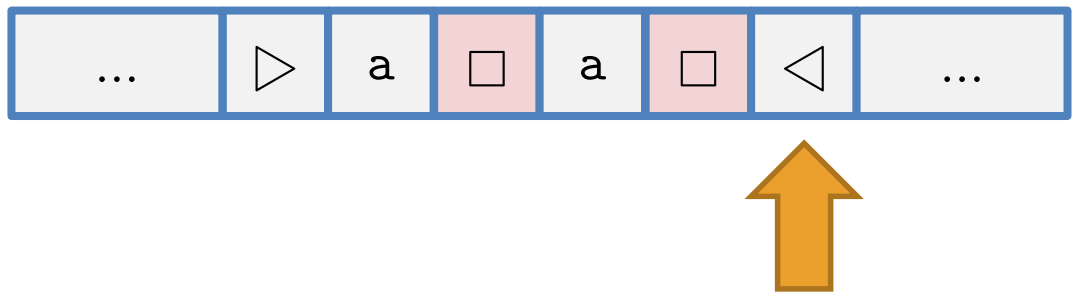
Step 1: Sweep from left to right and cross out every other a

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

Memory



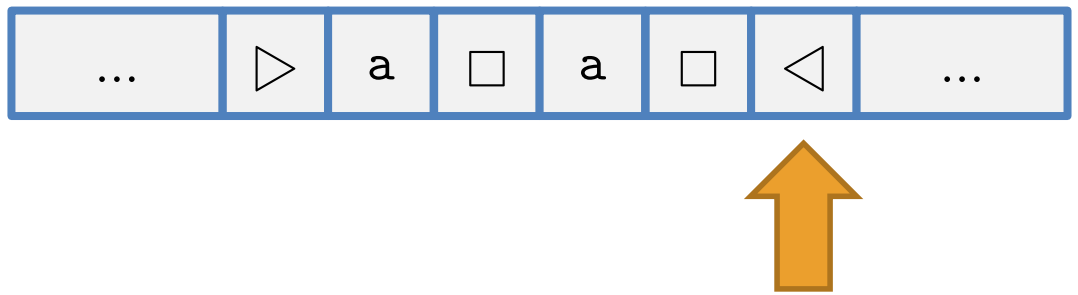
Step 1: Sweep from left to right and cross out every other a

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

Memory



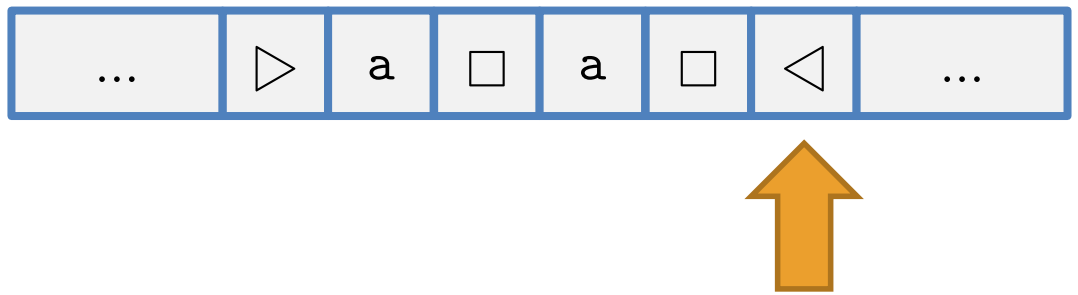
Step 2: Does the tape has only one a? NO

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

Memory

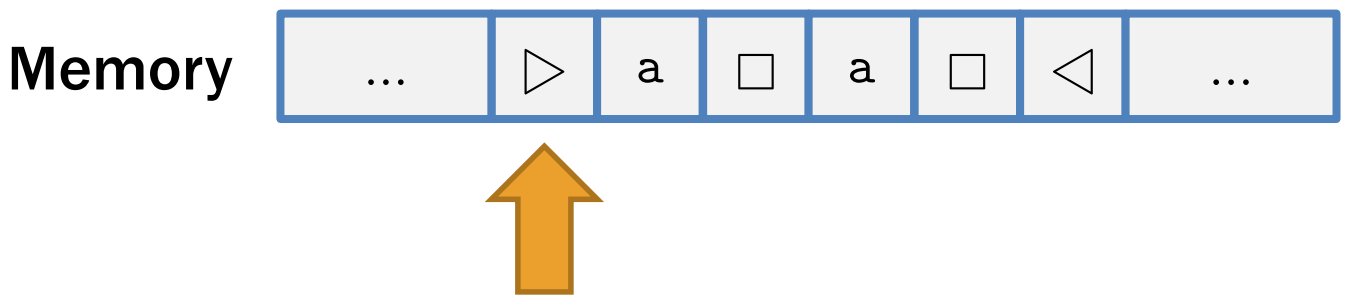


Step 3: Does the tape has an odd number of a's? NO

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.



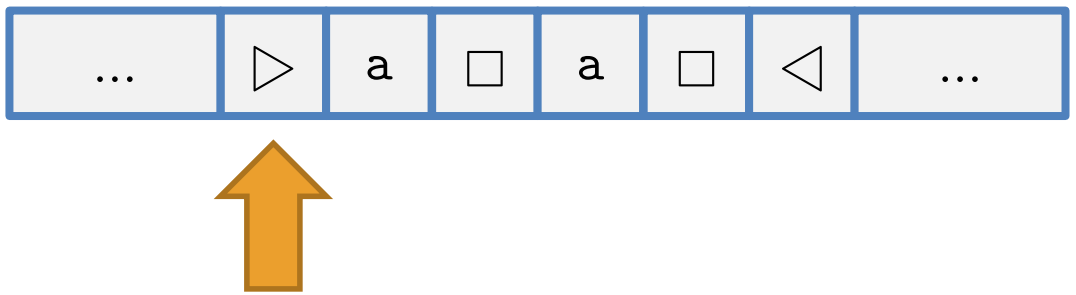
Step 4: Move back to the first cell

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

Memory

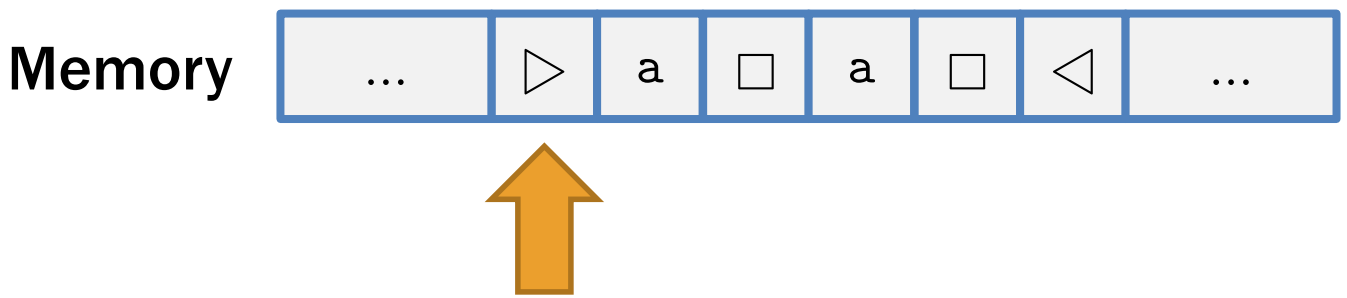


Step 5: Go back to step 1

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

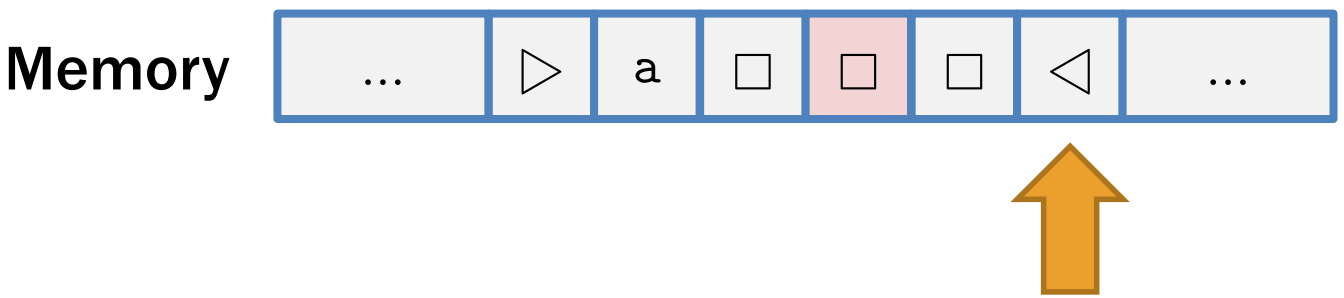


Step 1: Sweep from left to right and cross out every other a

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.



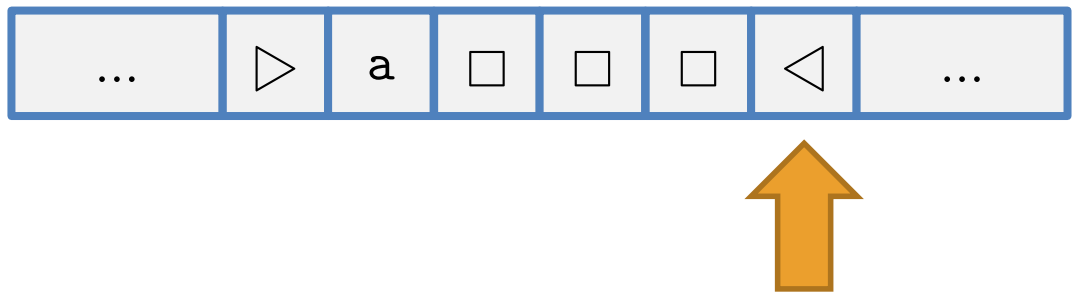
Step 1: Sweep from left to right and cross out every other a

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.

Memory

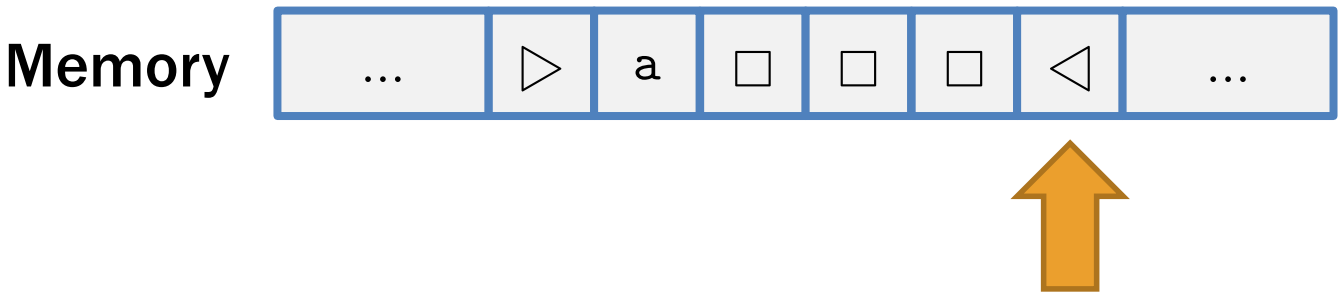


Step 2: Does the tape has only one a? YES

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- Pseudocode
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Intuition: We can check if $A = 2^n$ by dividing A with 2 until it results in an odd number. If the result is 1, then A is 2^n. Otherwise, A is not 2^n.



ACCEPT

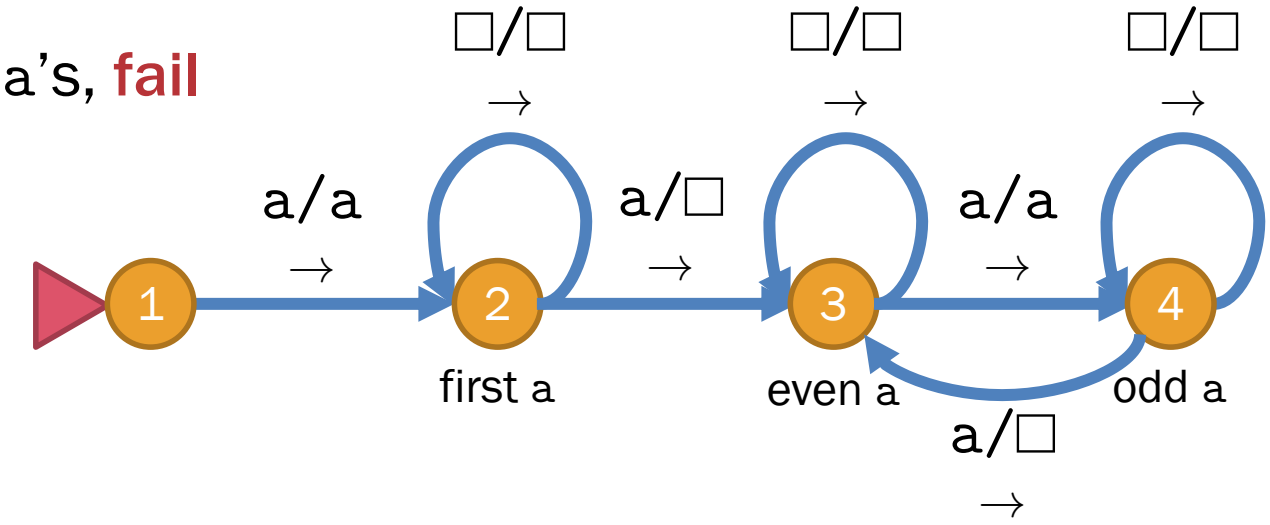
Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- From pseudocode to Turing machine
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1

Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

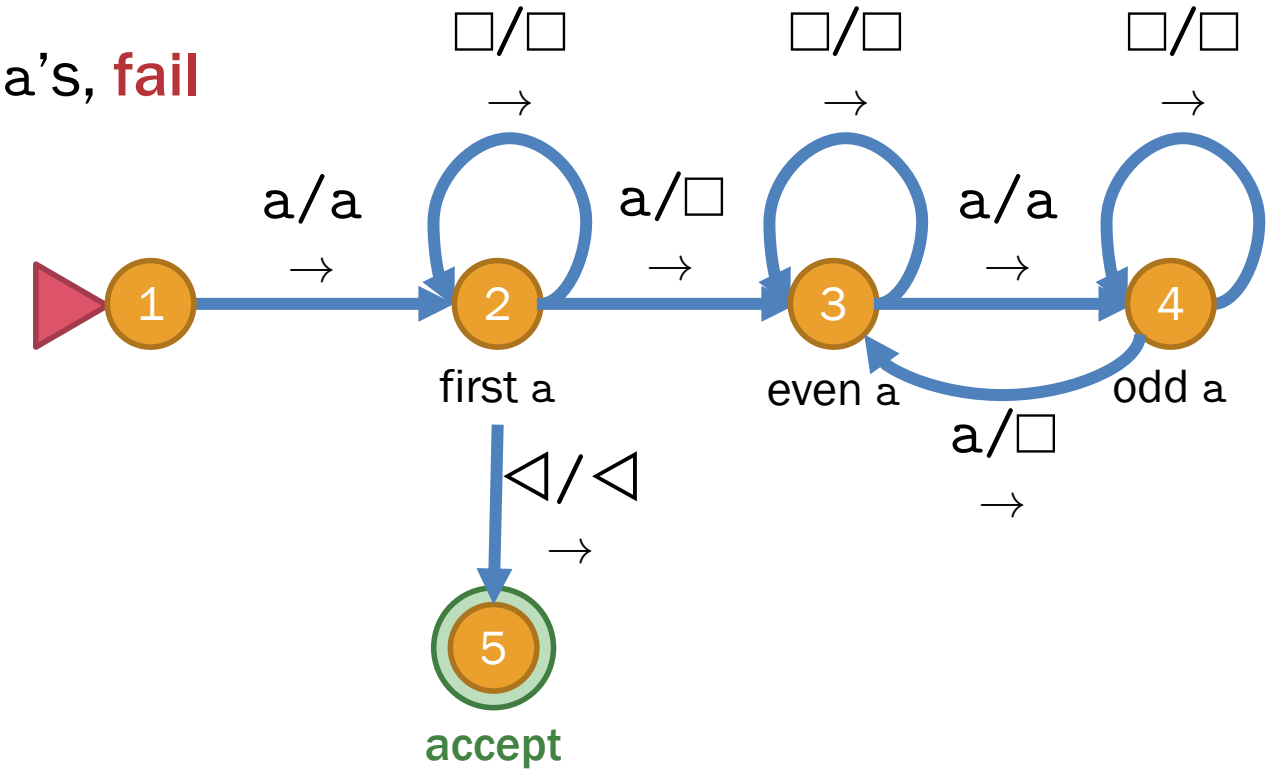
- From pseudocode to Turing machine

1. Sweep from left to right, cross out every other a
2. If the tape has only one a, **accept**
3. If the tape has an odd number of a's, **fail**
4. Move back to the first cell
5. Go back to step 1



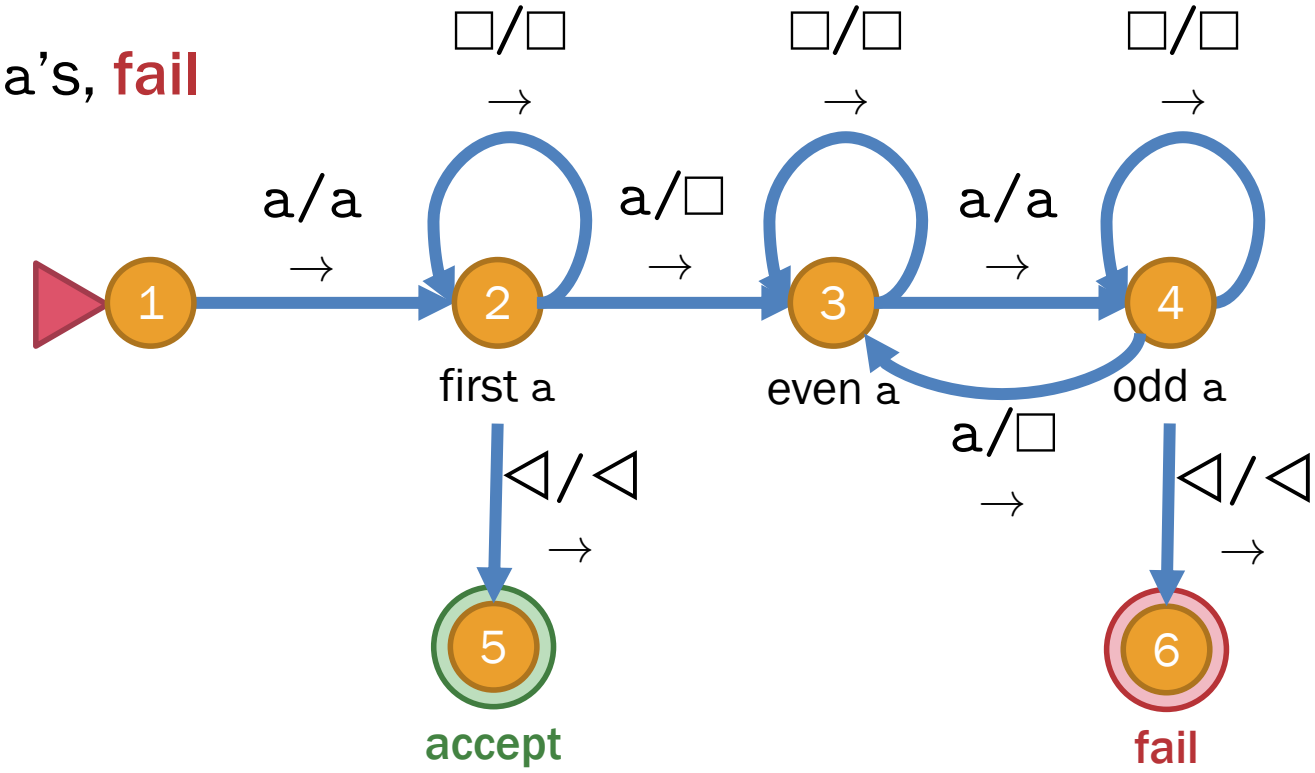
Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- From pseudocode to Turing machine
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1



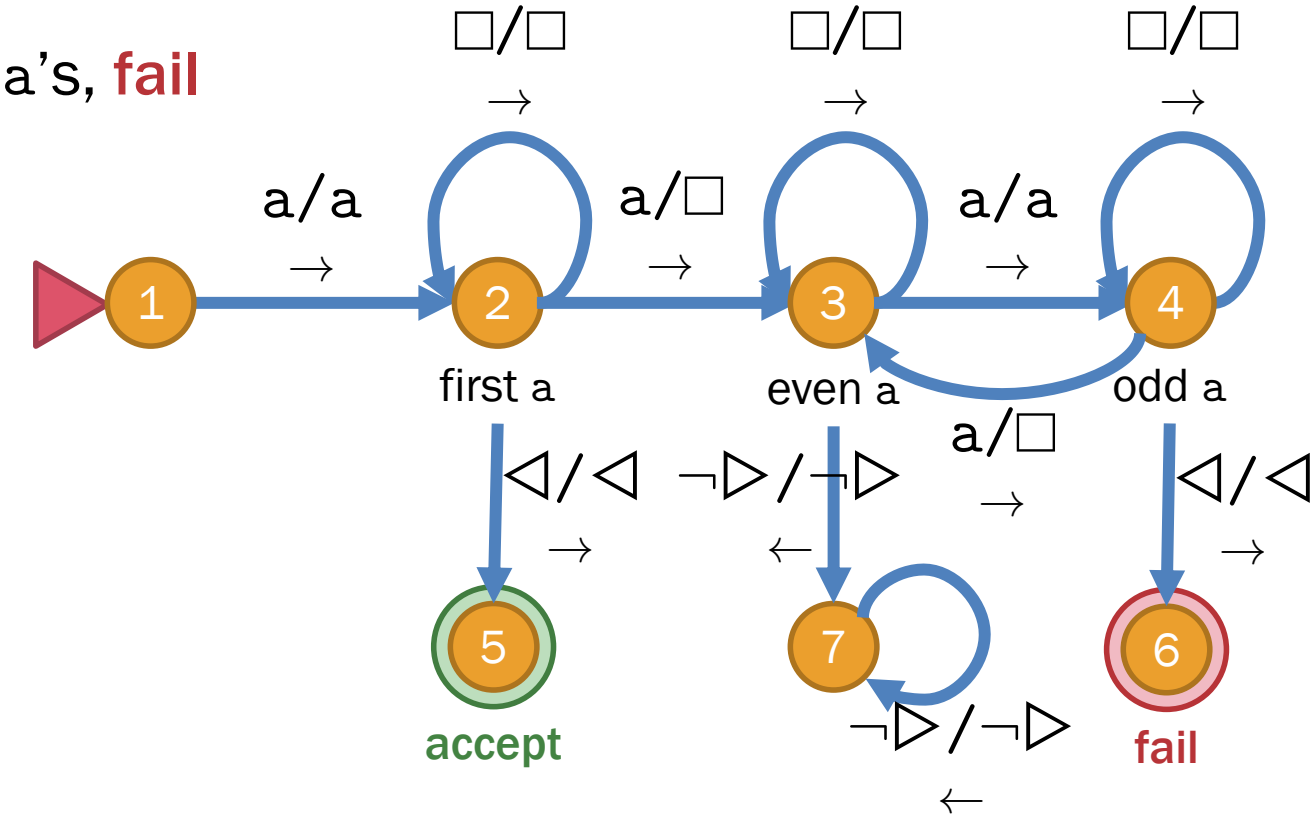
Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- From pseudocode to Turing machine
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1



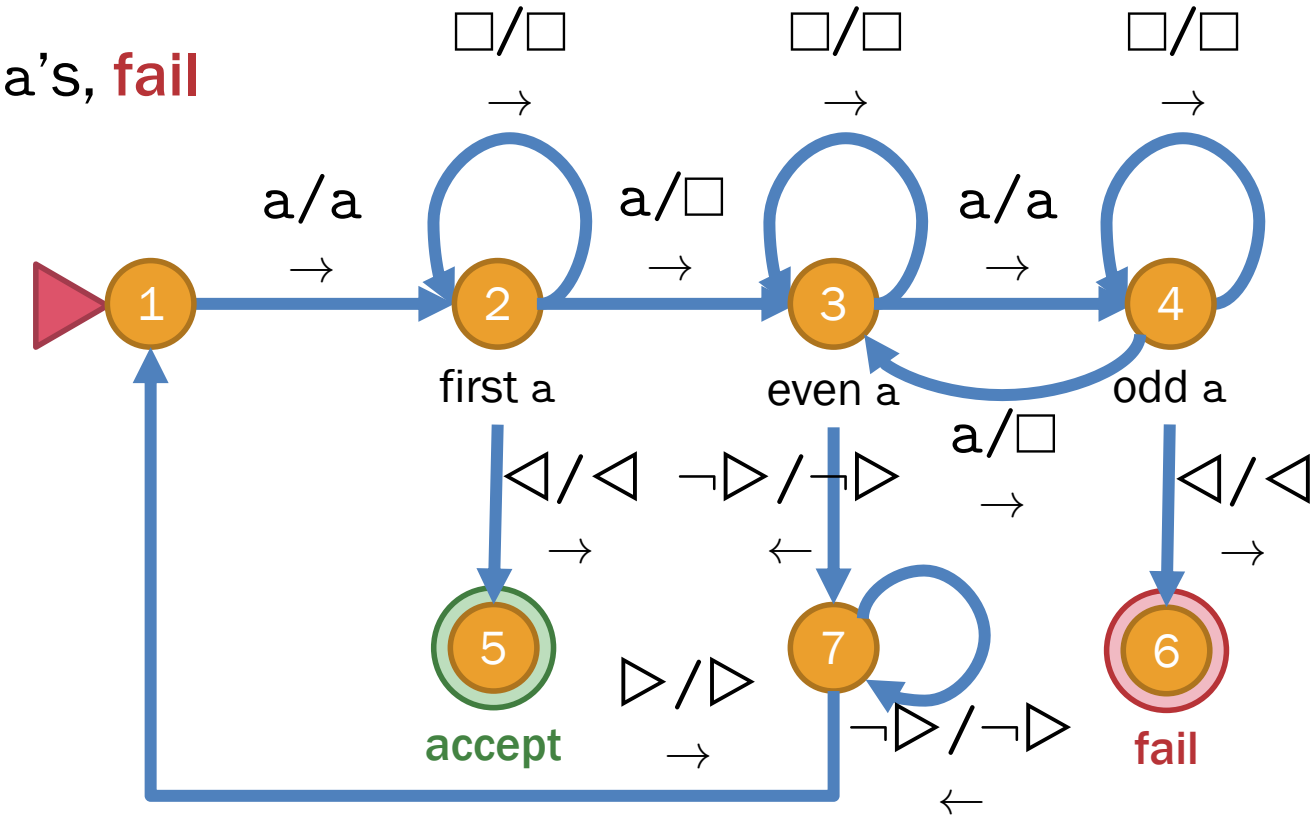
Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- From pseudocode to Turing machine
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1



Turing Machine for $L_{\text{expo}} = \{\mathtt{a}^{2^n} \mid n > 0\}$

- From pseudocode to Turing machine
  1. Sweep from left to right, cross out every other a
  2. If the tape has only one a, **accept**
  3. If the tape has an odd number of a's, **fail**
  4. Move back to the first cell
  5. Go back to step 1



4. Conclusion

Conclusion

- Esoteric programming is Turing-complete
  - It can be used to recognize any recursively enumerable language
  - Multiplication, power, exponential, and prime computation can be done
  - Some languages are practically used, while the others are designed to stretch the boundary of compiler design

Thank You

prachya.boonkwan@nectec.or.th

kaamanita@gmail.com