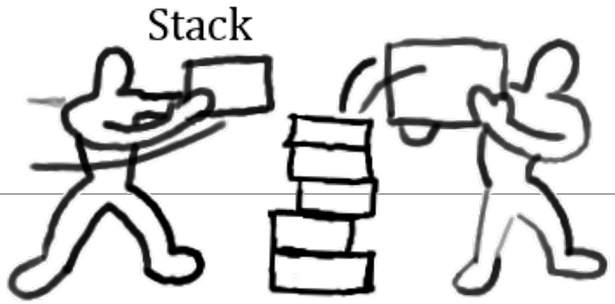
01418231
Data Structures

LECTURE-3-STACK AND APPLICATIONS



Powered by: Assit.Prof.Jirawan Charoensuk,Ph.d.

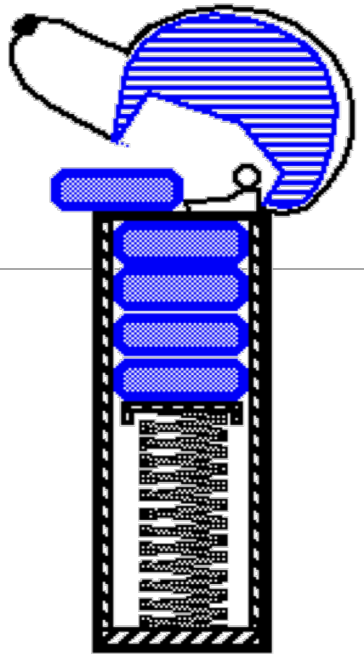
Reference: picture

Agenda

- What is Stack ?
- What is the properties of stack ?
- Applications of Stack
- Summary

https://th.aliexpress.com/item/33006540510.html

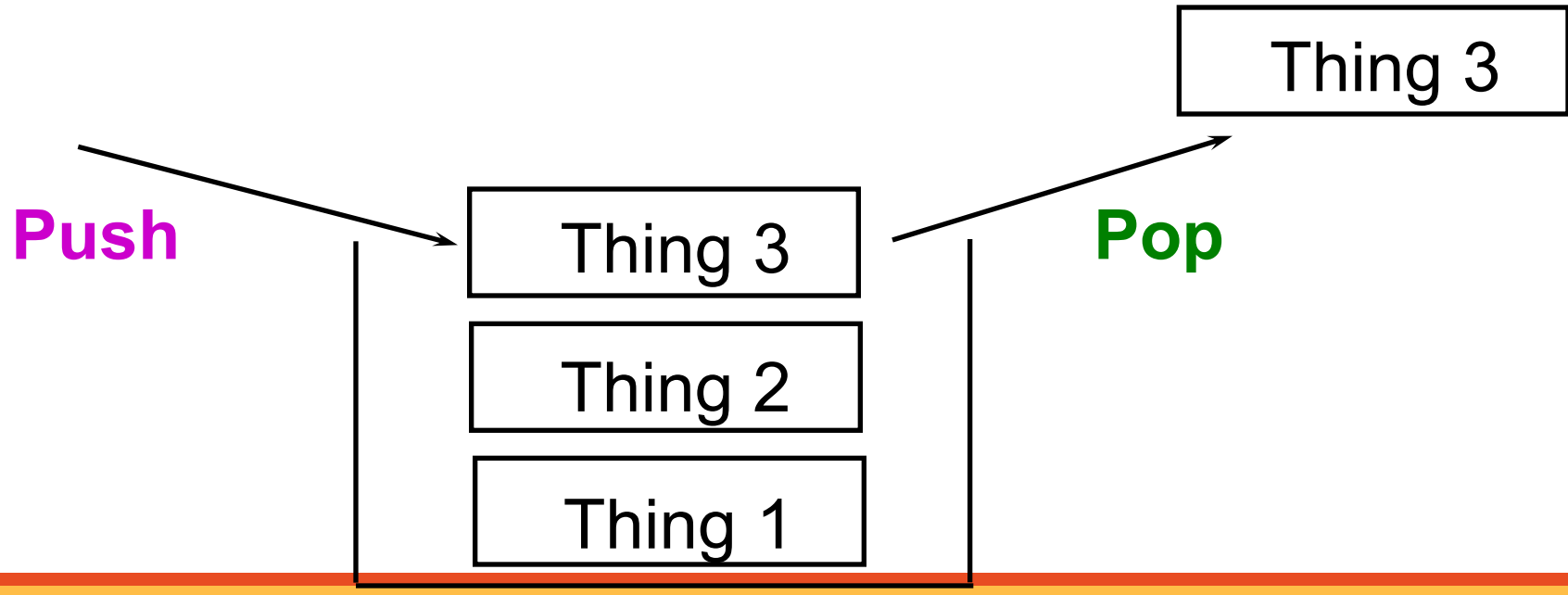
What is Stack ?



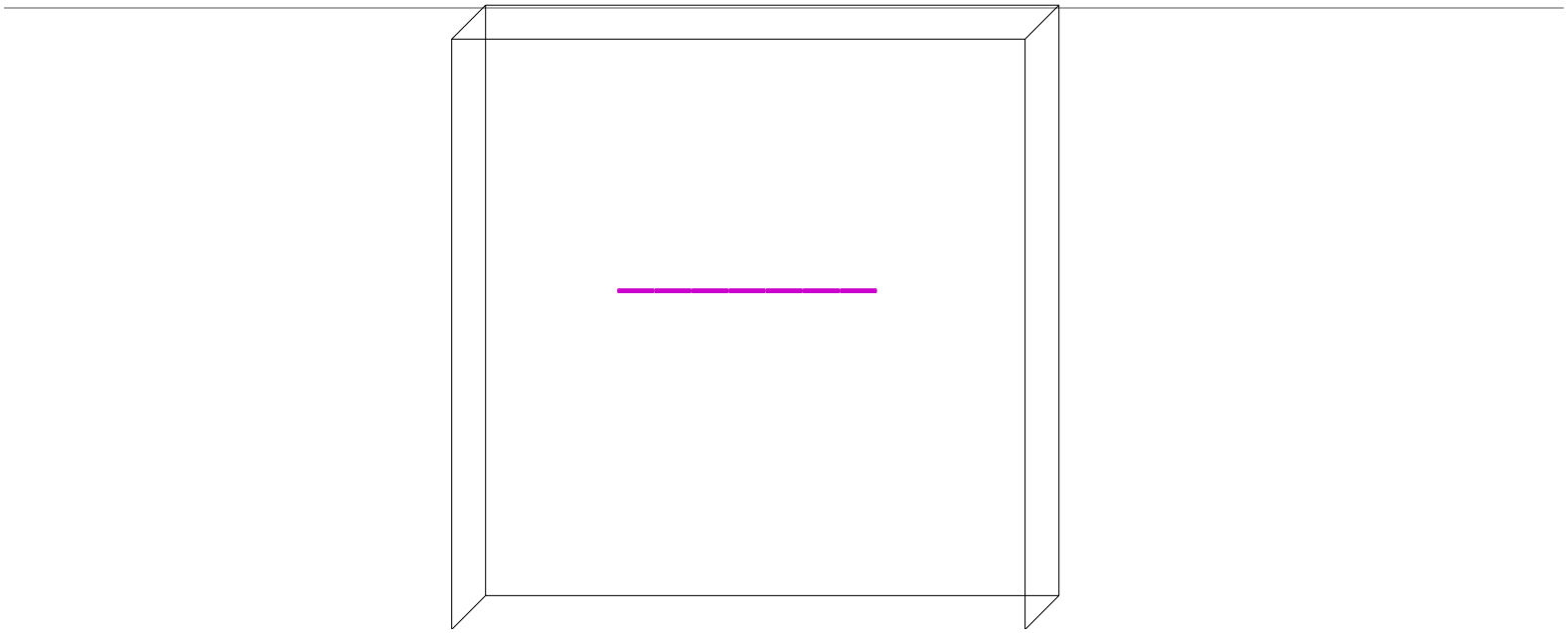
Stacks

A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle.

- Object can only be inserted to the top ("_____")
- Object can only be removed the top ("_____")

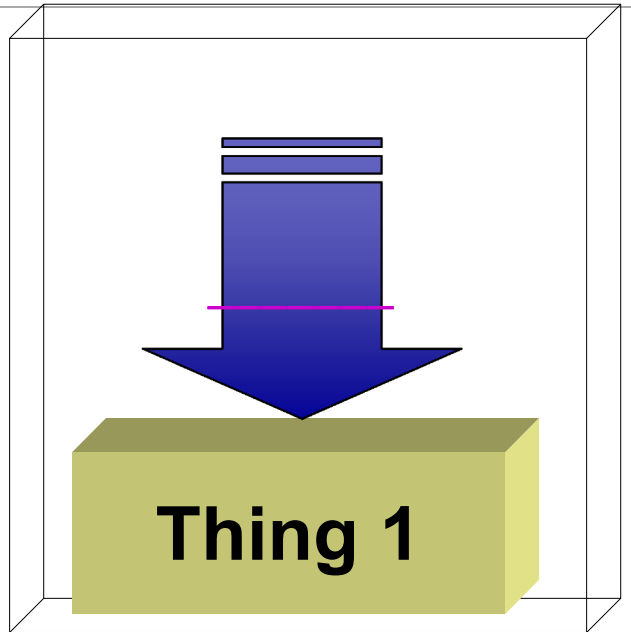


What is a Stack?



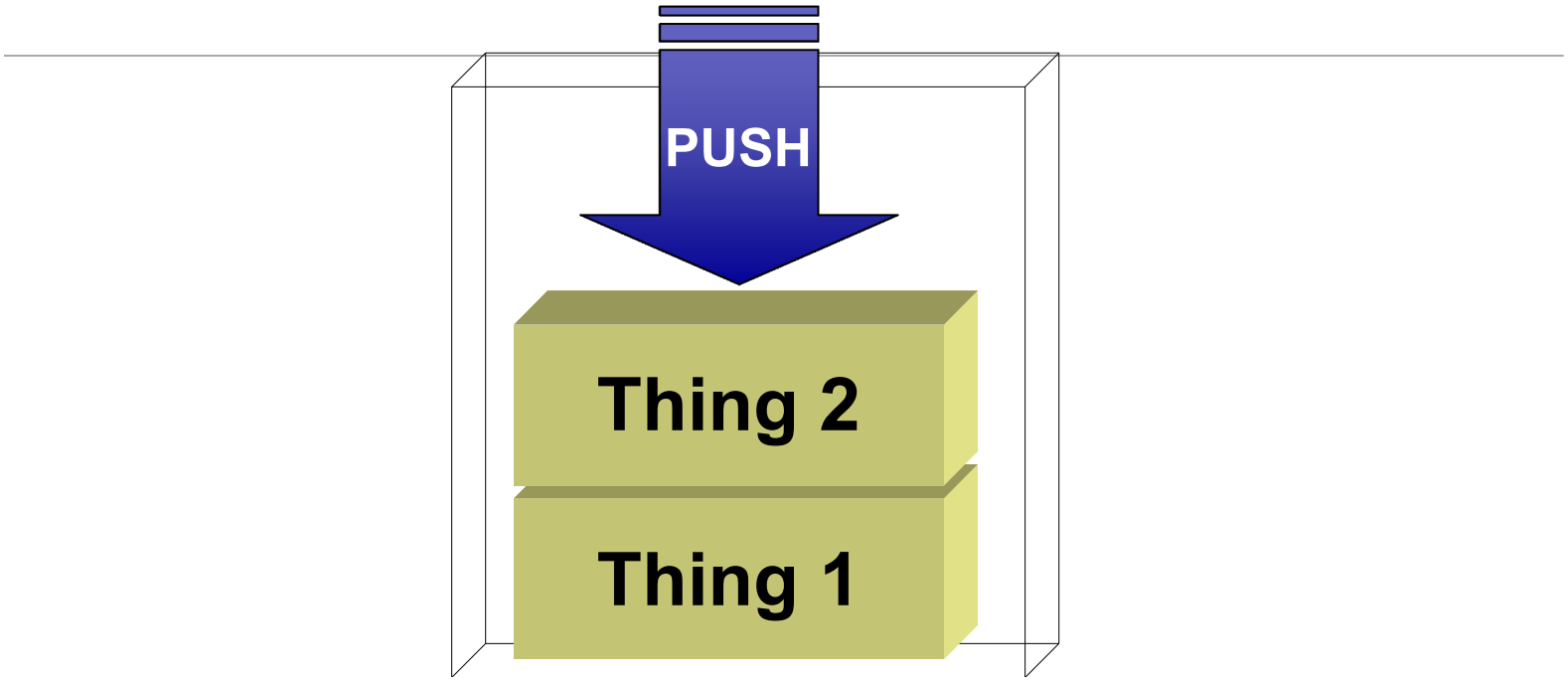
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



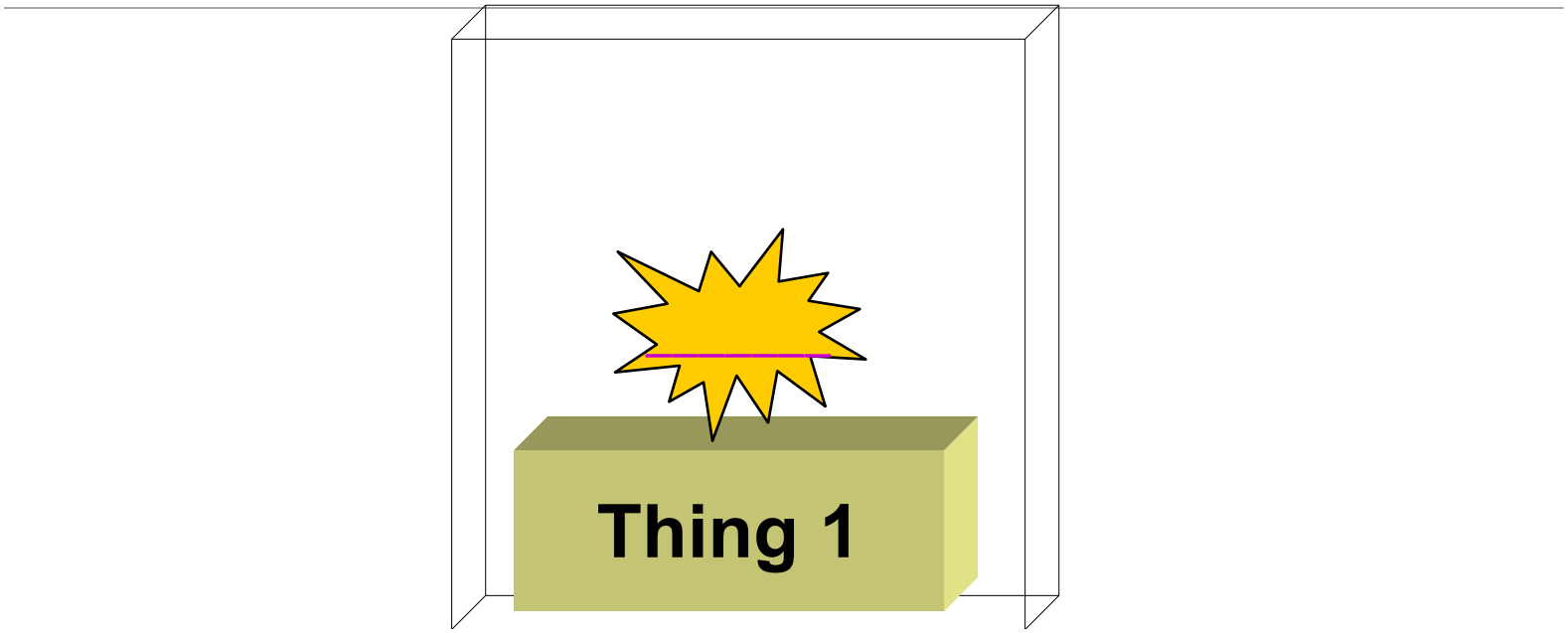
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



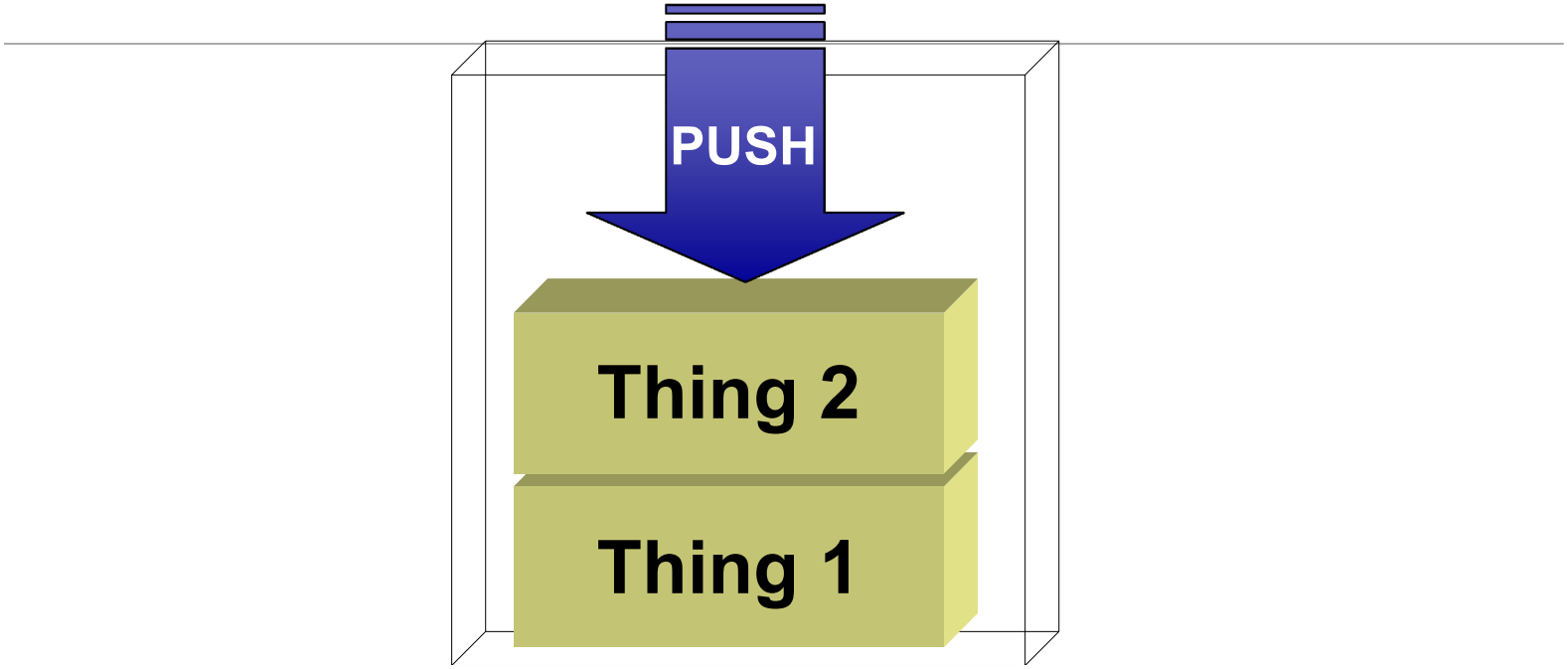
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



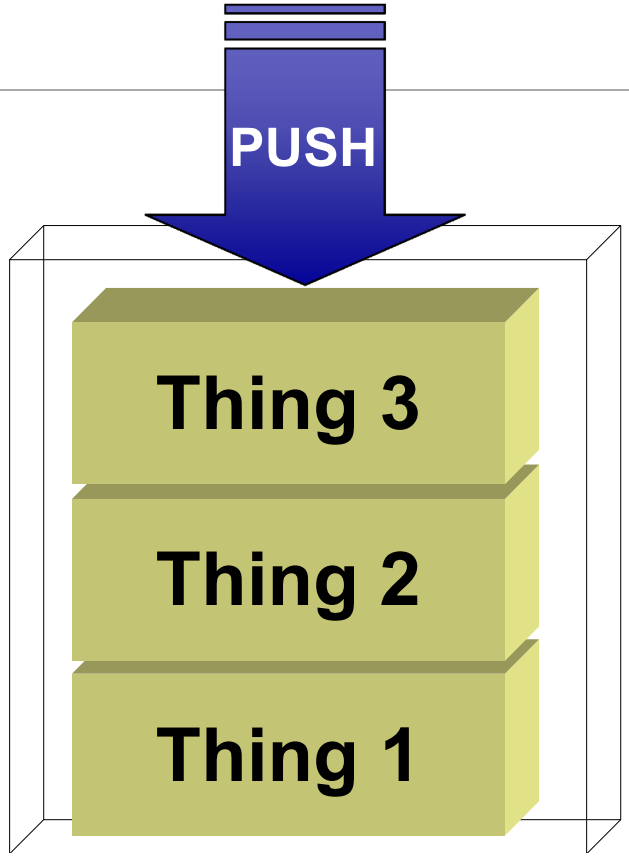
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



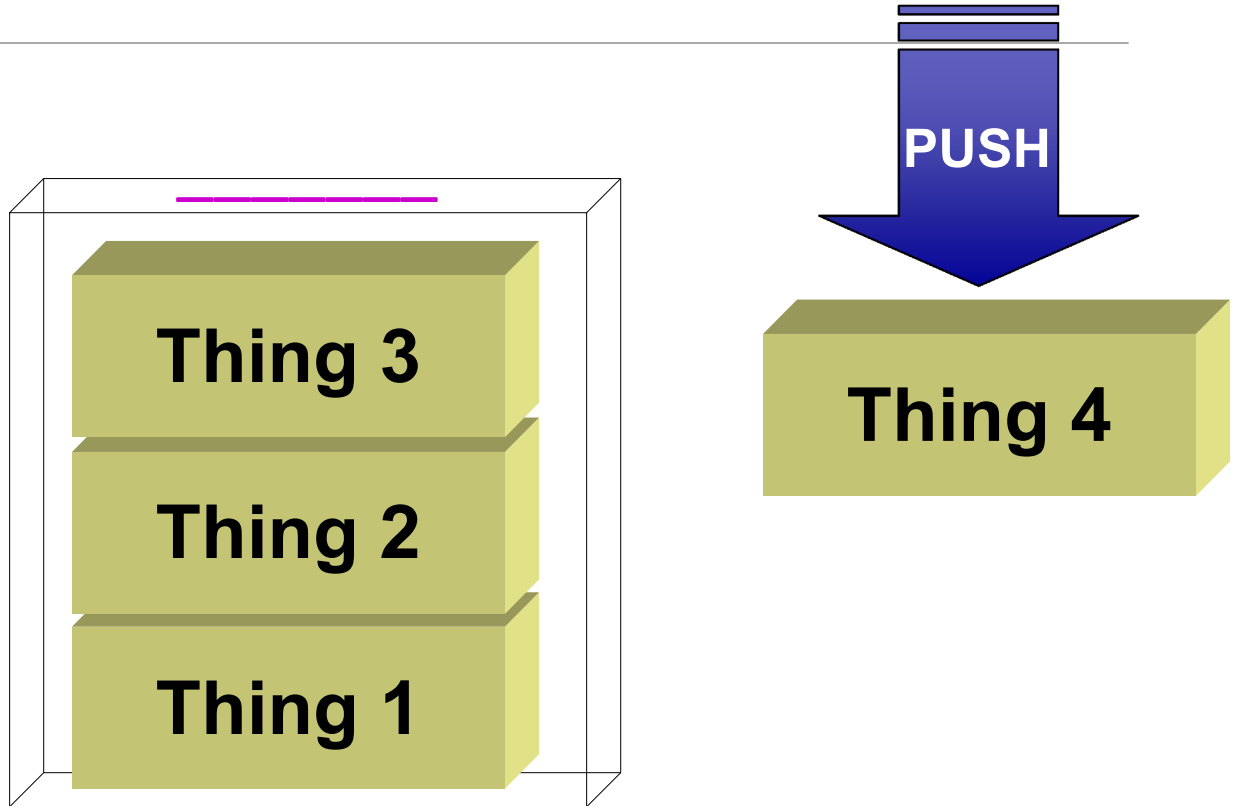
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



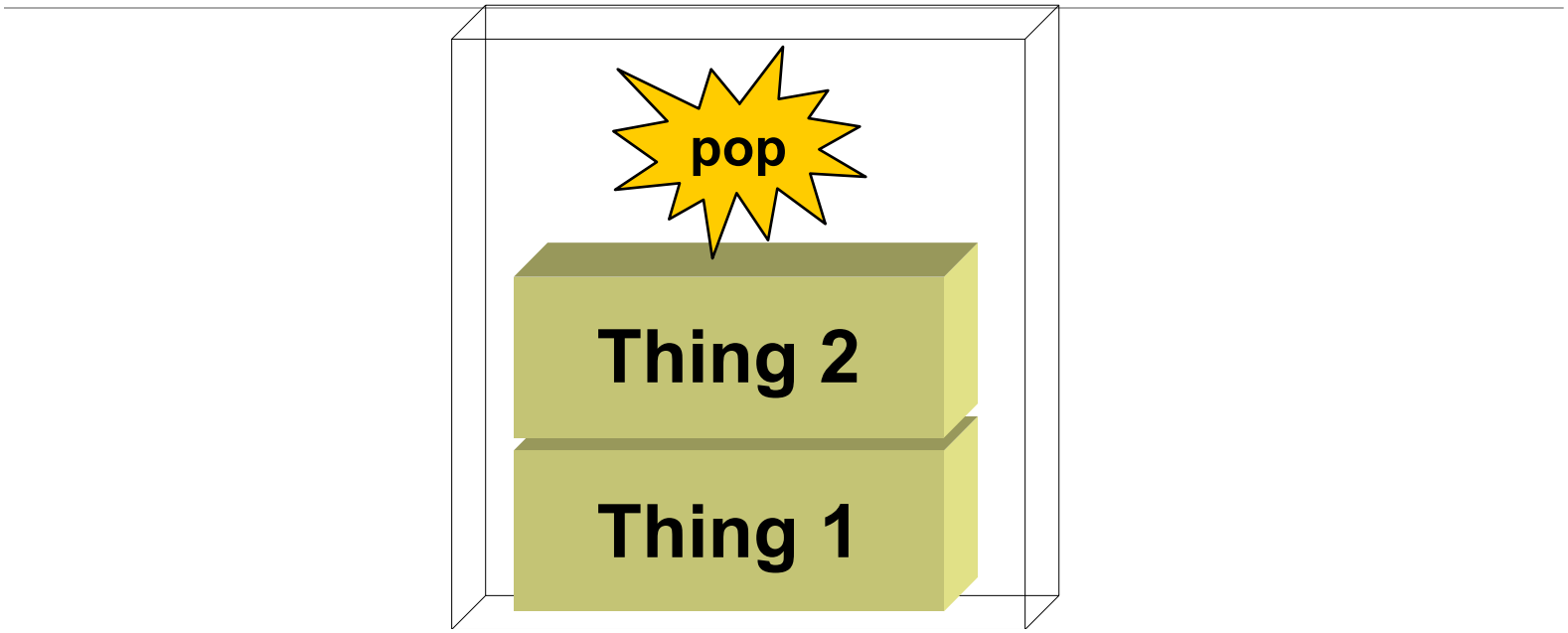
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



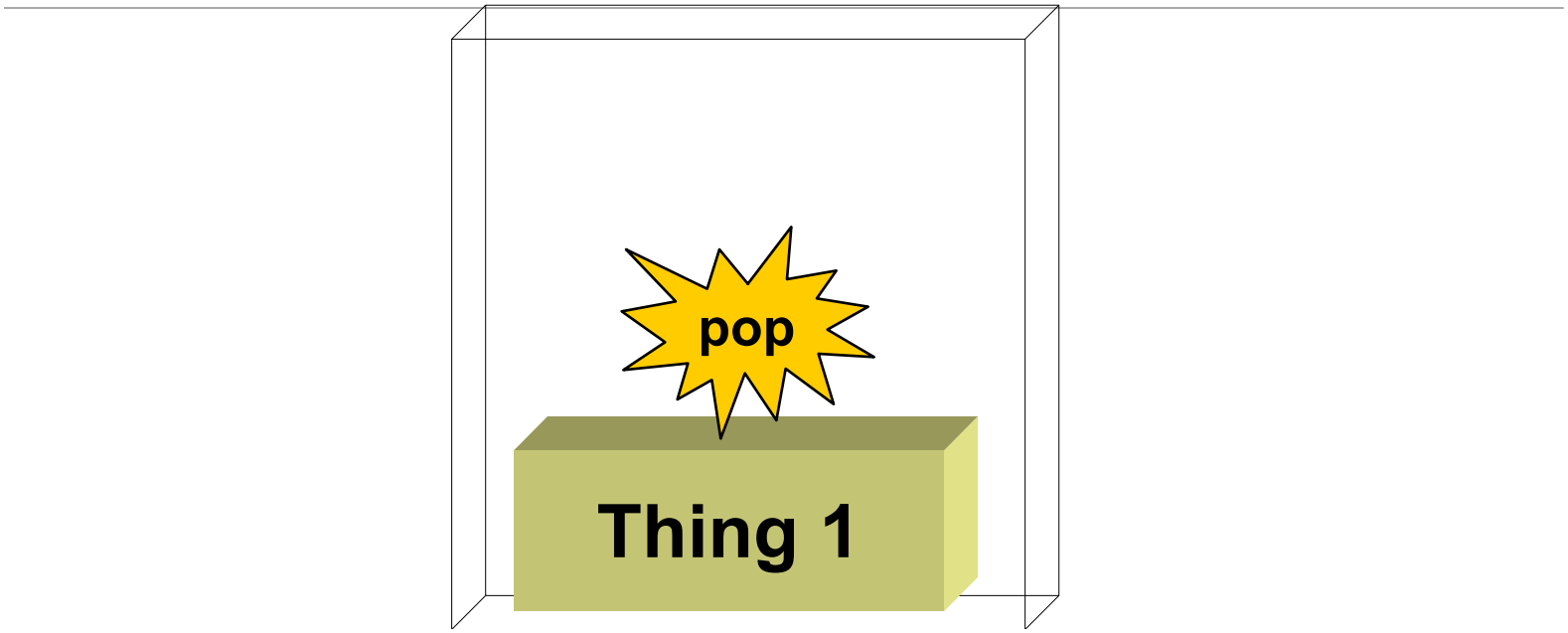
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



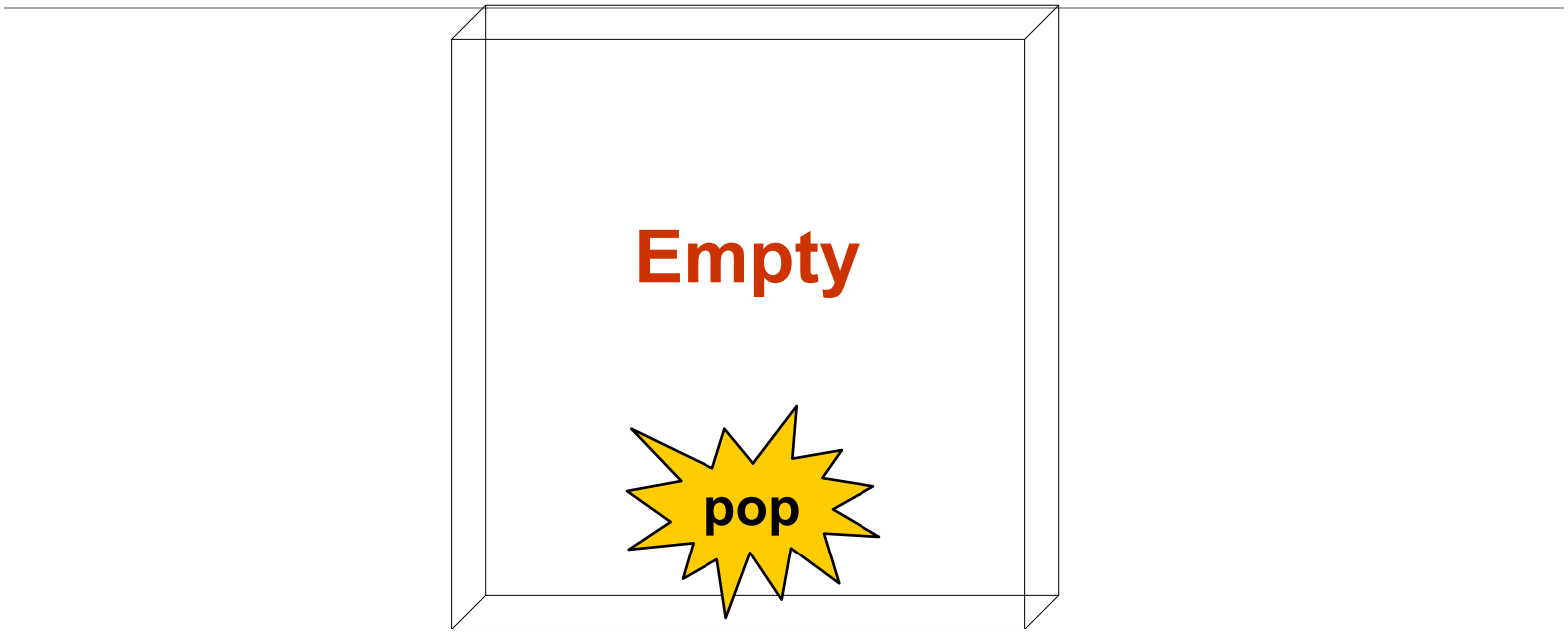
A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is a Stack?



A data structure for storing items which are to be accessed in last-in first-out order (LIFO).

What is the properties of stack ?

Properties

Idea: a “Last In, First Out” (LIFO) data structure

Function:

- _____: Add to top of stack
- _____: Remove from top of stack and return that top value
- _____: Return topmost item
- _____: is it full?
- _____: is it empty?
- _____: empty stack
- _____ : Return the number of object in stack

Stack

The Stack as a Logical Data Structure

The stack is an idea

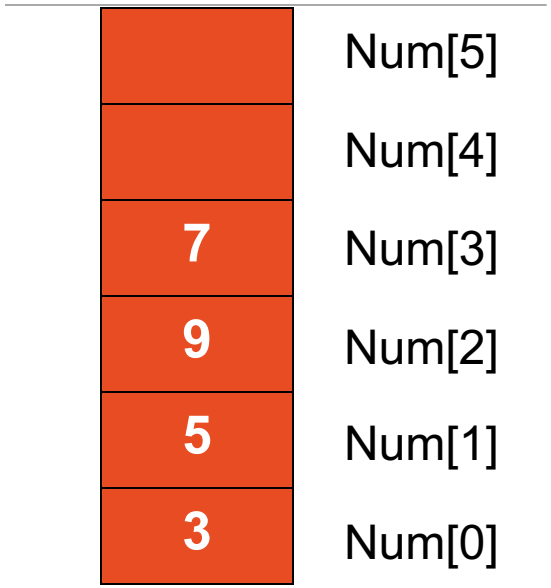
It implies a set of logical behaviors

It can be implemented various ways

- Static implement : array
- Dynamic implement: linked list

Stacks: Static Implementation

- A simple ways of implement ADT stack
- Add object from left to right
- Using index of array to calculate and refer to position or size of stack



- **Top =**
- **Size =**

Stacks: Static Implementation

The array and record storing

- Stack element may be "_____"
- Can't use "_____", if stack is full
- Must declare Top variable

Stack Operations using Array

Initial setting

Step 1: Define a constant **'SIZE'** of array.

Step 2: Declare all the **functions** used in stack implementation.

Step 3: Create a one dimensional array with fixed size (**int stack[SIZE]**)

Step 4: Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)

```
1.  #define SIZE 5

2.  void push(int);
3.  void pop();
4.  void display();

5.  int stack[SIZE],
6.  int top = -1;
```

http://btechsmartclass.com/DS/U2_T2.html

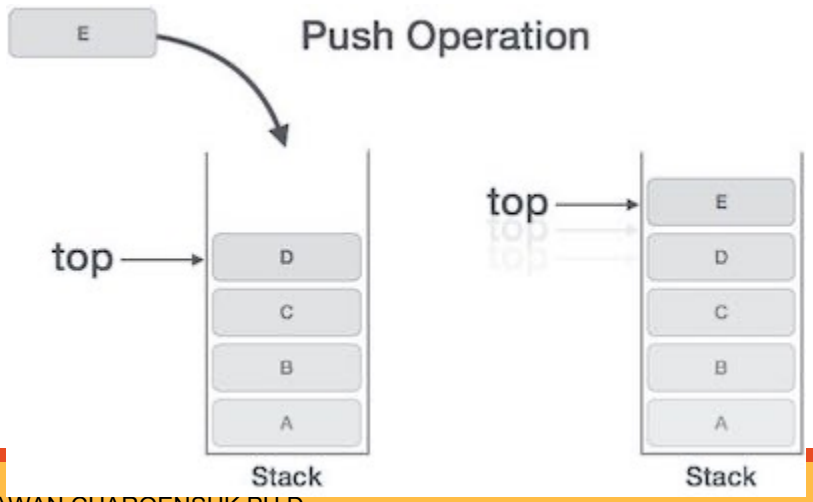
push(value) - Inserting value into the stack

Step 1: Check whether **stack** is **FULL**. (**top == SIZE-1**)

Step 2: If it is _____, then display **"Stack is FULL!!! "** and terminate the function.

Step 3: If it is _____, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

```
void push(int value){
  if(top == SIZE-1)
    printf("\nStack is Full!!! ");
  else{
    top++;
    stack[top] = value;
    printf("\nInsertion success!!!");
  }
}
```



https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

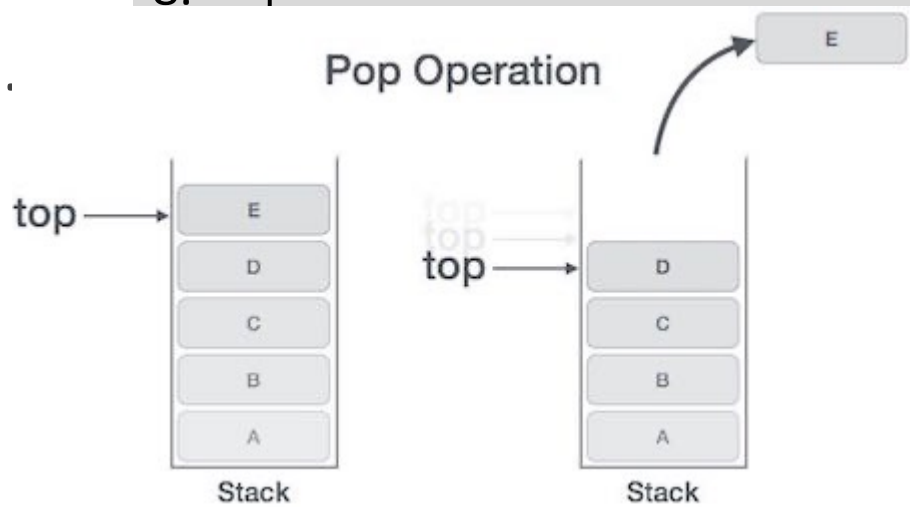
pop() - Delete a value from the Stack

Step 1: Check whether **stack** is **EMPTY**. (**top == -1**)

Step 2: If it is _____, then display **"Stack is EMPTY!!!"** and terminate the function.

Step 3: If it is _____, then delete **stack[top]** and decrement **top** value by one (**top--**).

```
void pop(){
  if(top == -1)
    printf("\nStack is Empty!!! ");
  else{
      printf("\nDeleted : %d",
      stack[top]);
    top--;
  }
}
```



https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

display() - Displays the elements of a Stack

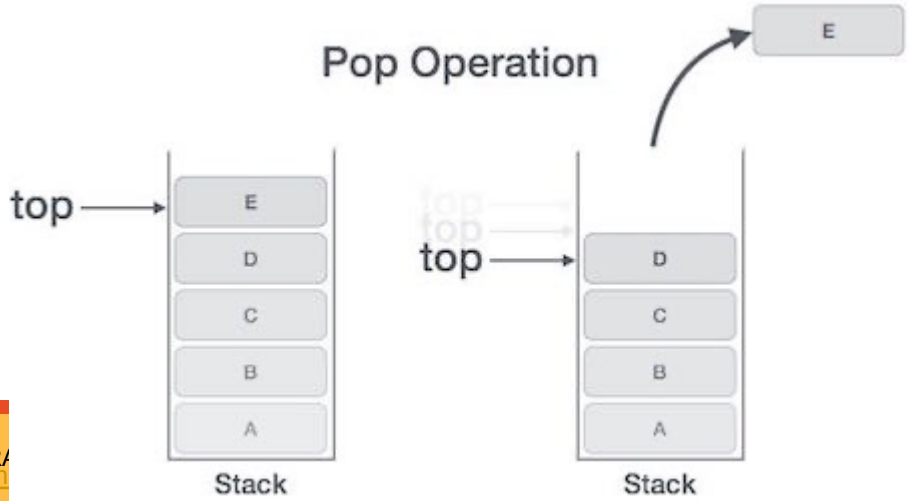
Step 1: Check whether **stack** is **EMPTY**. (**top == -1**)

Step 2: If it is _____, then display **"Stack is EMPTY!!!"** and terminate the function.

Step 3: If it is _____, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).

Step 3: Repeat above step until **i** value becomes '0'.

```
void display(){
  if(top == -1)
    printf("\nStack is Empty!!!");
  else{
    int i;
    printf("\nStack elements are:\n");
    for(i=top; i>=0; i--)
   printf("%d\n",stack[i]);
  }
}
```



https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

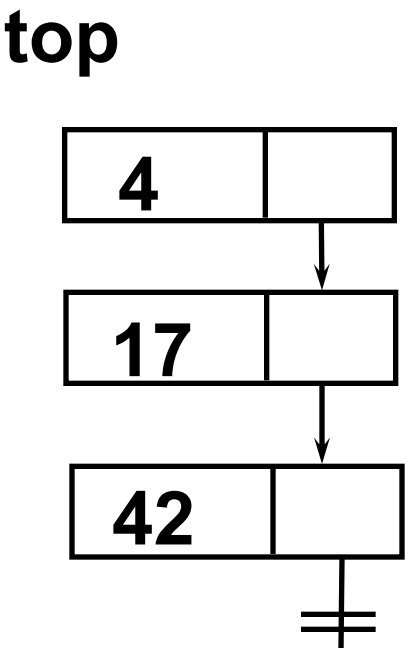
```
#include<stdio.h>
#include <stdlib.h>

#define SIZE 5

void push(int);
void pop();
void display();

int stack[SIZE], top = -1;
main()
{
  int value, choice;
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
      case 1: printf("Enter the value to be insert: ")
            scanf("%d",&value);
            push(value);display();
            break;
      case 2: pop(); display();
            break;
      case 3: display();
            break;
     case 4: exit(0);
      default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
```

```
30. void push(int value){
31.    if(top == SIZE-1)
32.       printf("\nStack is Full!!!\n\n");
33.    else{
34.       top++;
35.       stack[top] = value;
36.       printf("\nInsertion success!!!\n\n");
37.    }
38. }
39. void pop(){
40.    if(top == -1)
41.       printf("\nStack is Empty!!!\n\n");
42.    else{
43.       printf("\nDeleted : %d", stack[top]);
44.       top--;
45.    }
46. }
```

```
47. void display(){
48.    if(top == -1)
49.       printf("\nStack is Empty!!!\n\n");
50.    else{
51.       int i;
52.       printf("\nStack elements are:\n\n");
53.       for(i=top; i>=0; i--)
54.         printf("%d\n",stack[i]);
55.    }
56. }
```

Stacks: Dynamic Implementation

A linked list with restricted set of operations to change its state: only modified from one to end



Defining the Node Type

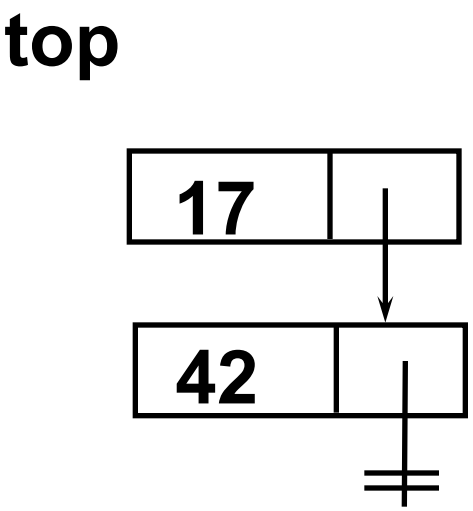
This is the simple data structure we will use in the following example

```
Node defines a record
  data isoftype Num
  next isoftype Ptr
endrecord  // Node
```

```
struct node
{
  int data;
  struct node *next;
};
```

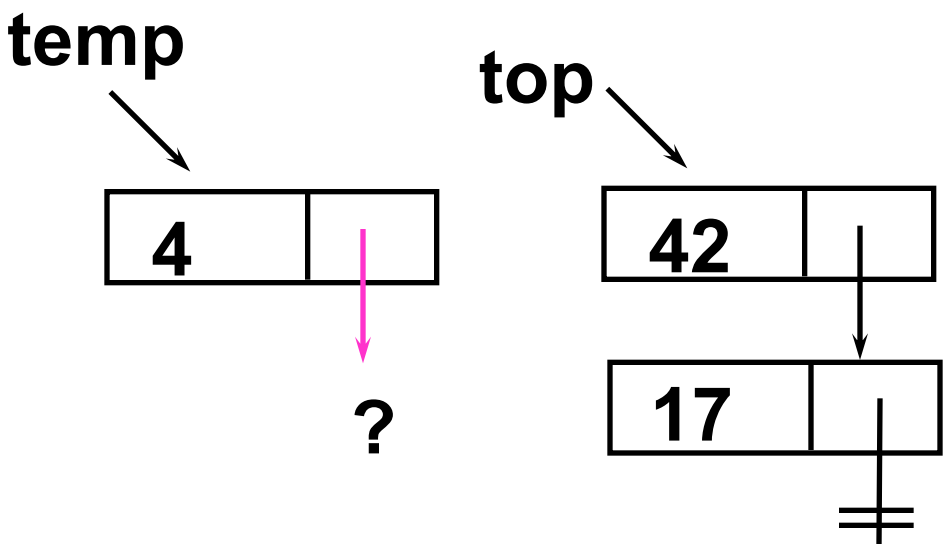
Push

1. Create new node
2. Add it to the front



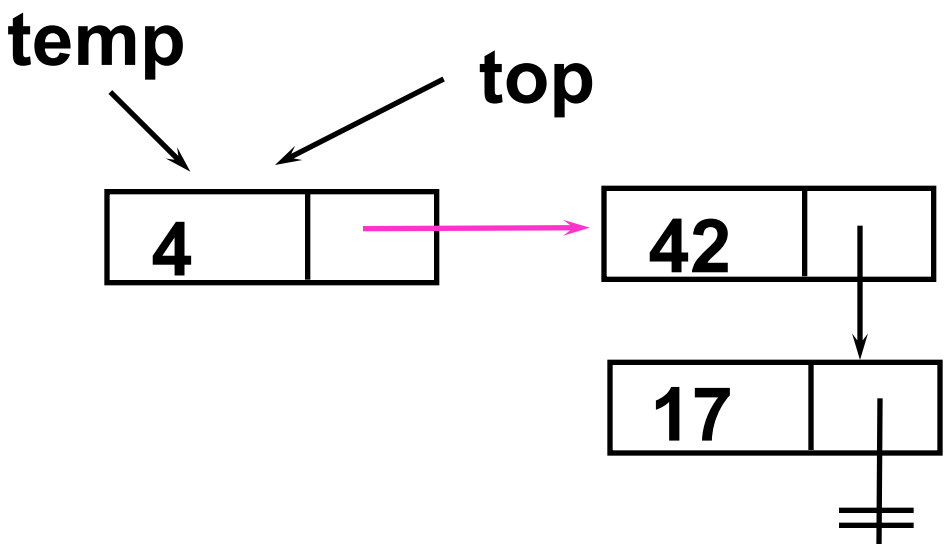
Push

1. Create new node
2. Add it to the front



Push

1. Create new node
2. Add it to the front

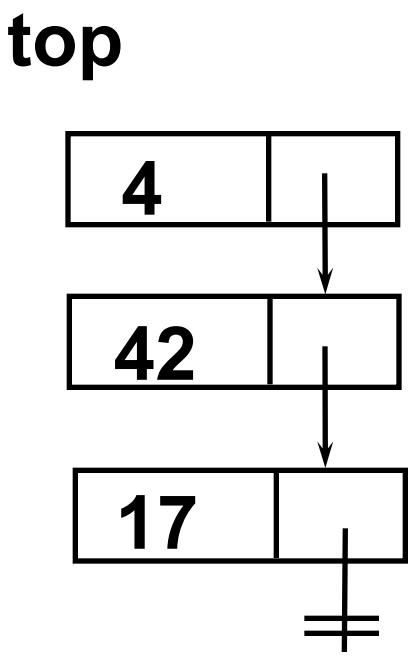


Push

```
Procedure Push (value isoftype in Num,
    top isoftype in/out Ptr)
  // Push one value onto stack
  temp isoftype Ptr
  temp = newNode
  temp->data = value
  temp->next = top
  top = temp
endprocedure // Push
```

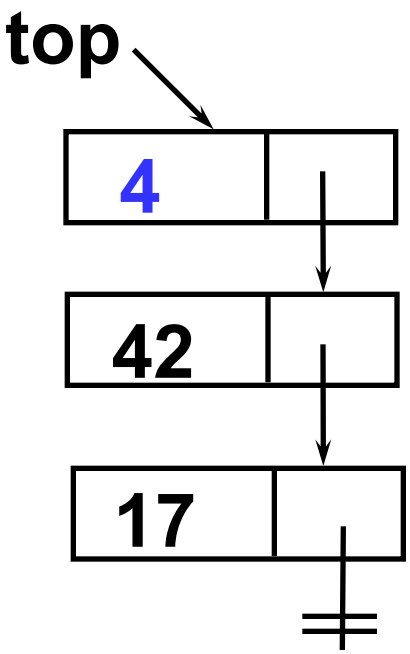
Pop

1. Capture the first value (to return)
2. Remove the first node (move top to next)



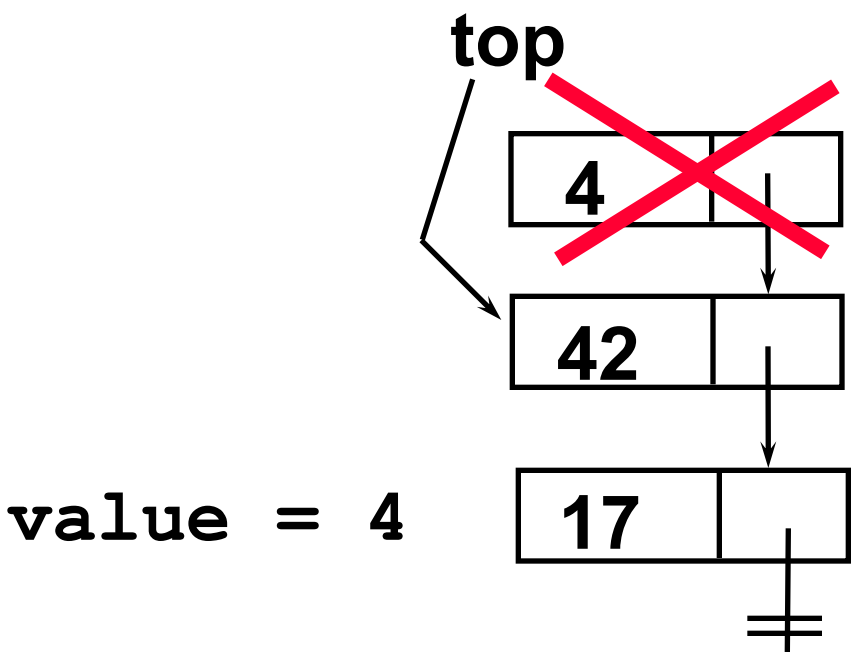
Pop

1. Capture the first value (to return)
2. Remove the first node (move top to next)



Pop

1. Capture the first value (to return)
2. Remove the first node (move top to next)



Pop

```
Procedure Pop (value isoftype out Num, top isoftype in/out Ptr ,
    result isoftype out Boolean)  // Pop an element off the stack
  if(top == NULL) then
    result = FALSE
  else
    result = TRUE
    value = top->data
    top = top->next
  endif
End procedure  // Pop
```

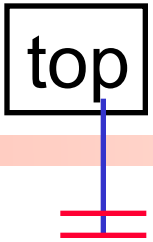
Algorithm Fragment

```
...
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
  print(N)
endif
```

```
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
...
```

```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

OK =
N =

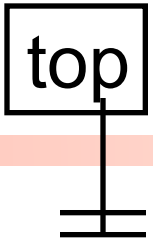


```
Procedure Push
    (value isoftype in Num,
     top isoftype in/out Ptr)
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
  print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

temp =

OK =
N =

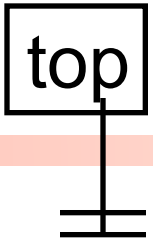


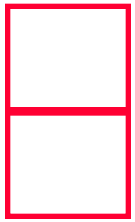
```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr)
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

temp =

OK =
N =





```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr)
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

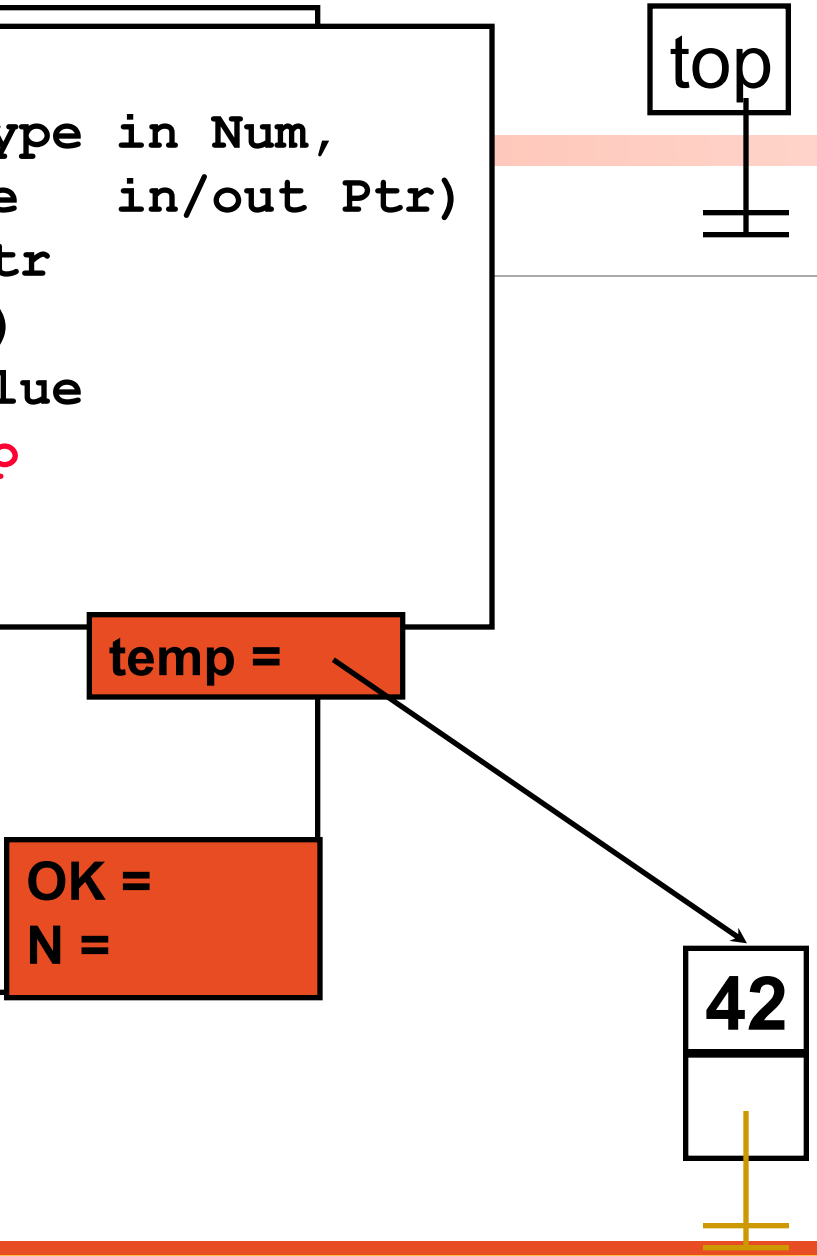
```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

temp =

OK =
N =

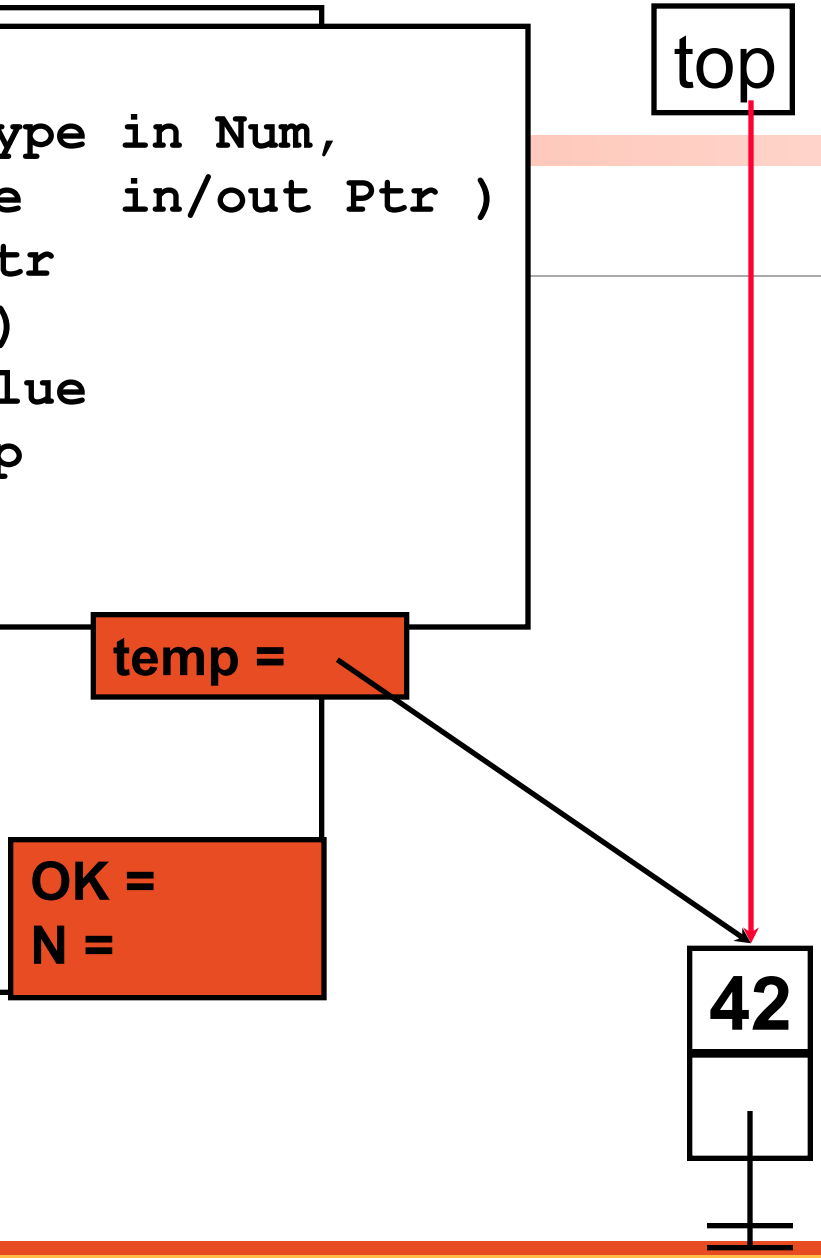
```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr)
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```



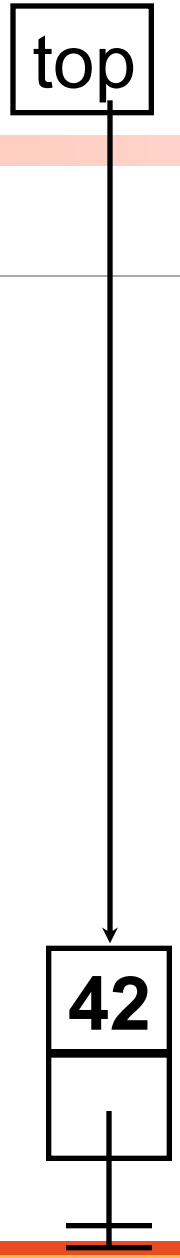
```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr )
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```



```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

OK =
N =

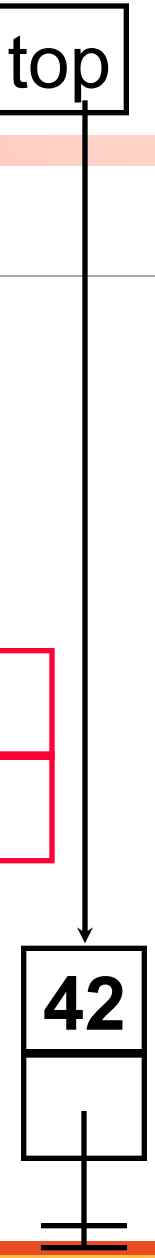


```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr )
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

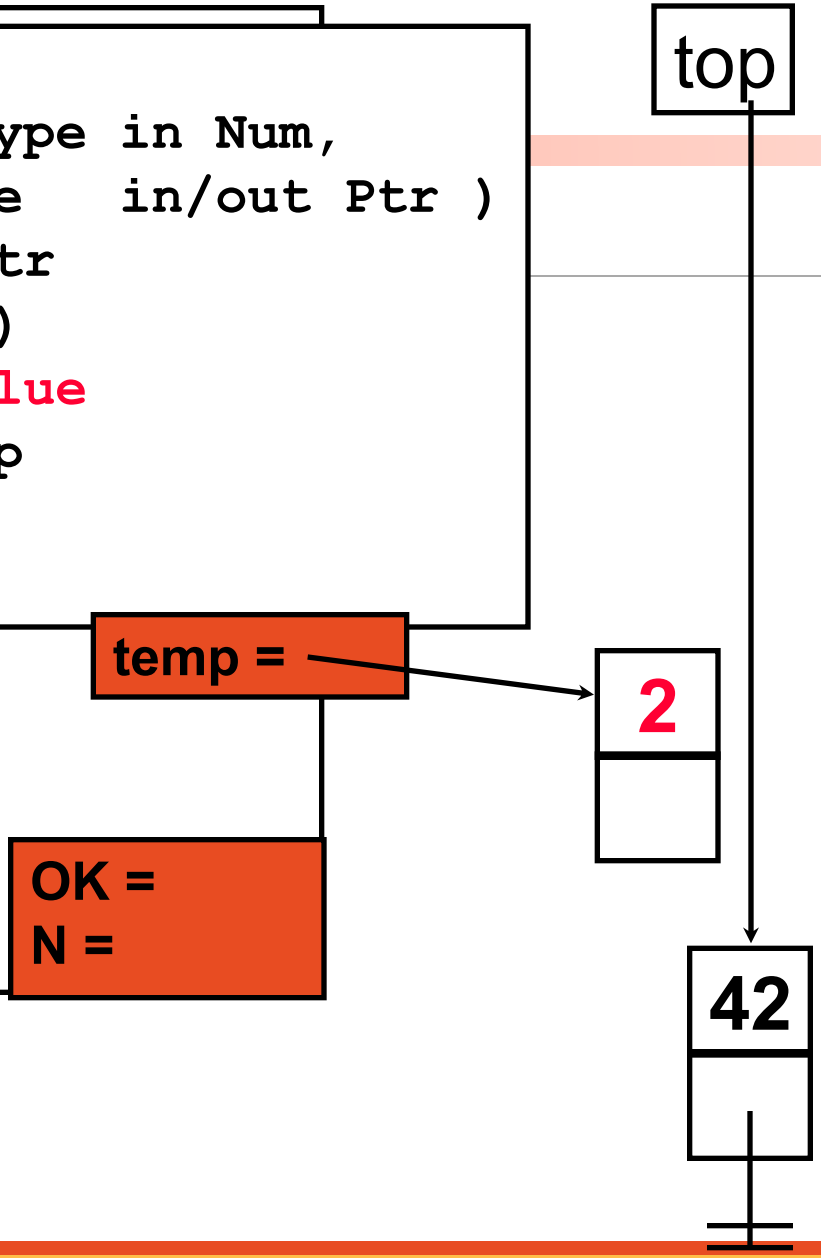
temp =

OK =
N =



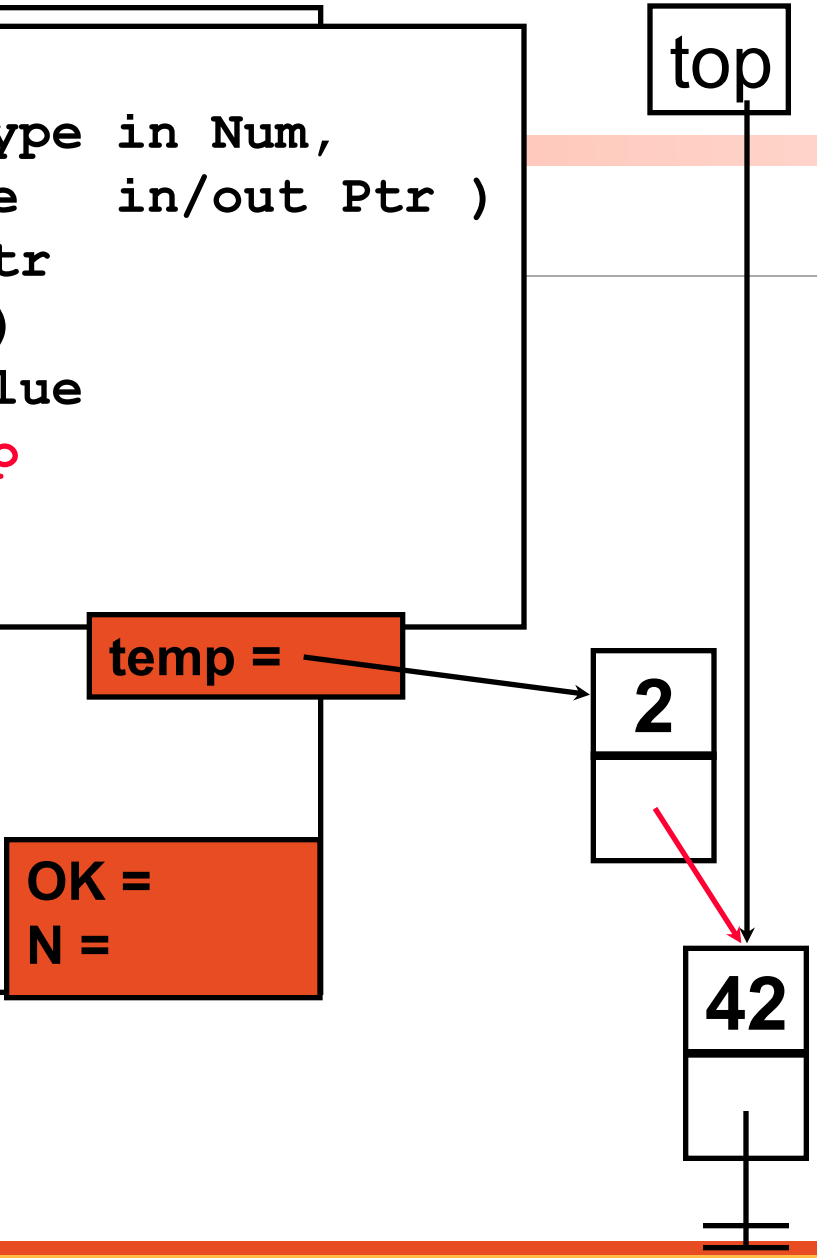
```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr )
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```



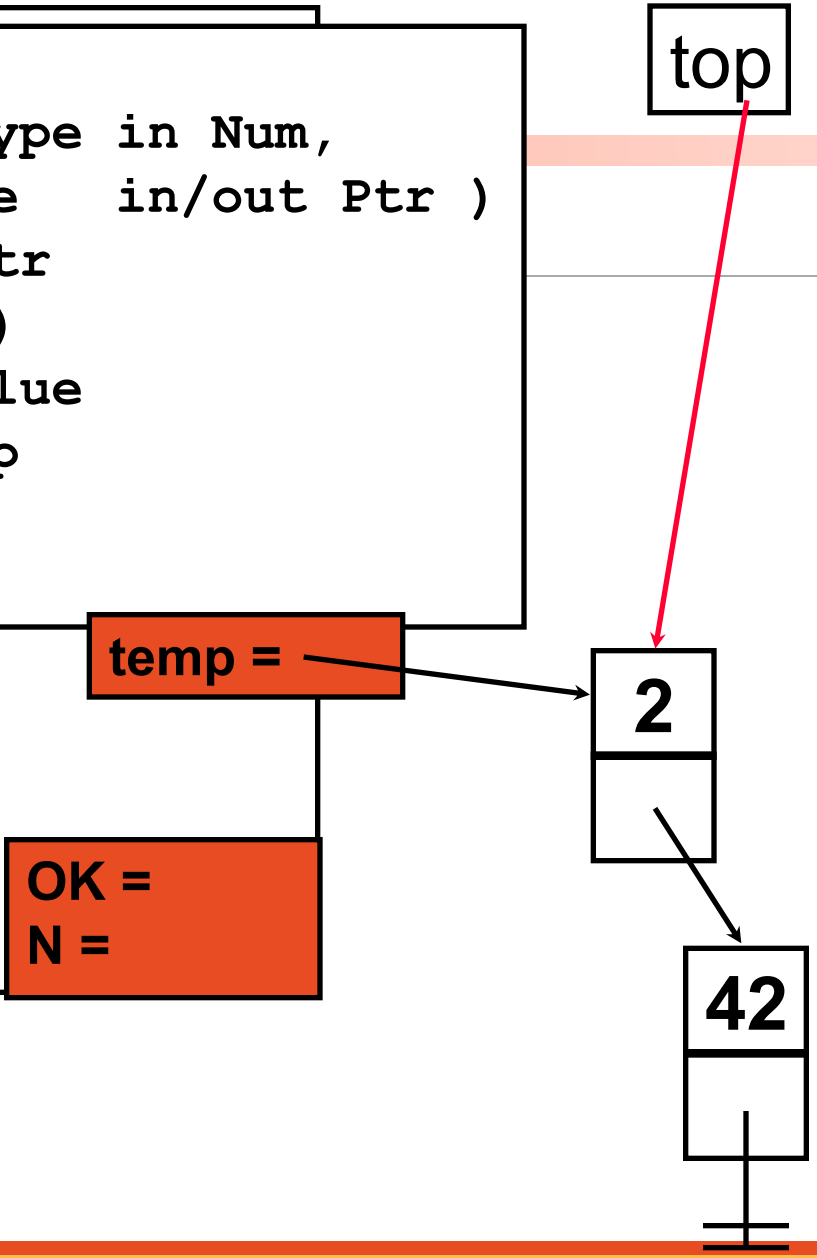
```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr )
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```



```
Procedure Push
     (value isoftype in Num,
      top isoftype   in/out Ptr )
  temp isoftype Ptr
  temp = new(Node)
  temp->data = value
  temp->next = top
  top = temp
endprocedure
```

```
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

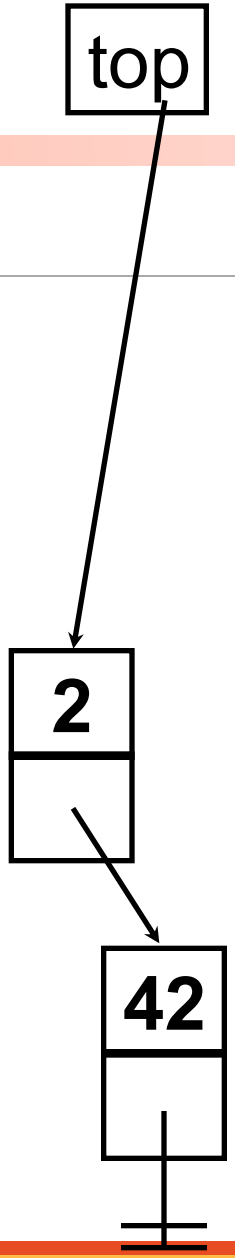


```
Procedure Pop
  value isoftype out Num,
  top isoftype   in/out Ptr,
  result isoftype out Boolean)
 if(top = NULL) then
  result = FALSE
 else
  result = TRUE
  value = top->data
  top = top->next
 endif
endprocedure
```

```
Pop(N, top, OK)
.
```

value =
result =

OK =
N =

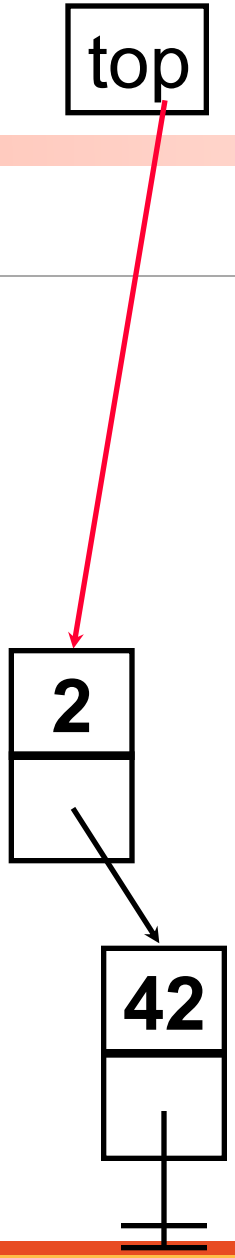


```
Procedure Pop
  value isoftype out Num,
  top isoftype   in/out Ptr,
  result isoftype out Boolean)
 if(top = NULL) then
  result = FALSE
 else
  result = TRUE
  value = top->data
  top = top->next
 endif
endprocedure
```

```
Pop(N, top, OK)
.
```

value =
result =

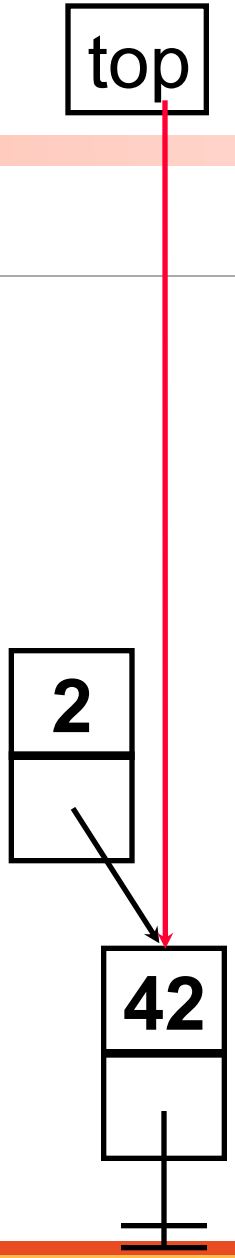
OK =
N =



```
Procedure Pop
   value isoftype out Num,
   top isoftype   in/out Ptr,
   result isoftype out Boolean)
  if(top = NULL) then
   result = FALSE
  else
   result = TRUE
   value = top->data
   top = top->next
  endif
endprocedure
Pop(N, top, OK)
.
```

value = 2
result = T

OK =
N =

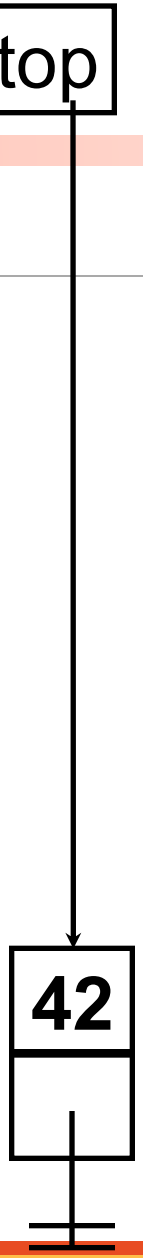


```
Procedure Pop
  value isoftype out Num,
  top isoftype   in/out Ptr,
  result isoftype out Boolean)
 if(top = NULL) then
  result = FALSE
 else
  result = TRUE
  value = top->data
  top = top->next
 endif
endprocedure
```

```
Pop(N, top, OK)
.
```

value = 2
result = T

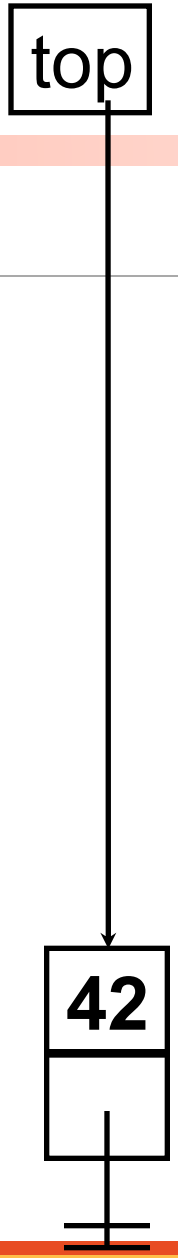
OK =
N =



```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

OK = T
N = 2

2



```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

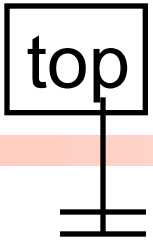
OK = T
N = 2

```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

OK = T
N = 7

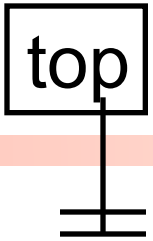
```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

OK = T
N = 42



```
.
top isoftype Ptr
OK isoftype Boolean
N isoftype Num
top = NULL
Push(42, top)
Push(2, top)
Pop(N, top, OK)
if(OK) then
 print(N)
endif
Push(7, top)
Pop(N, top, OK)
Pop(N, top, OK)
.
```

OK = T
N = 42



```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int data;
  struct node *next;
}*top = NULL;

void push(int);
void pop();
void display();
```

```
main()
{
  int choice, value;
  printf("\n:: Stack using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
        case 1: printf("Enter the value to be insert: ");
                scanf("%d", &value);
                push(value);display();
                break;
        case 2: pop(); display();break;
        case 3: display(); break;
        case 4: exit(0);
        default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
}
```

```
void push(int value)
{
    struct node *newnode;
    newnode = malloc(sizeof(struct node));

    newnode->data = value;
    if(top == NULL)
        newnode->next = NULL;
    else
        newnode->next = top;
    top = newnode;
    printf("\nInsertion is Success!!!\n");
}
```

```
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct node *temp = top;
    printf("\nDeleted element: %d\n", temp->data);
    top = temp->next;
    free(temp);
  }
}
```

```
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct node *temp = top;
    while(temp->next != NULL){
      printf("%d--->",temp->data);
      temp = temp -> next;
    }
    printf("%d--->NULL\n\n",temp->data);
  }
}
```

Summary

Abstract data type (ADT) is composed of

- A collection of data
- A set of operations on that data

The ADT stack operations have a last-in, first-out (LIFO) behavior

Stack can be implemented by arrays or linked lists

Summary

The ADT stack operations have a last-in, first-out (LIFO) behavior

Stack can be implemented by arrays or linked lists

Function:

- Push: Add to top of stack
- Pop: Remove from top of stack and return that top value
- Top: Return topmost item
- Is_Full: is it full?
- Is_Empty: is it empty?
- Initialize: empty stack
- Size_of_object : Return the number of object in stack

01418231 Data Structures

STACK APPLICATIONS

Agenda

Application areas use stacks:

- Bracket Checker
- Convert Infix to Postfix
- Evaluation of arithmetic expression
- Call Nested Procedures

Agenda

Application areas use stacks:

- Bracket Checker {[(.....)]}
- Convert Infix to Postfix
- Evaluation of arithmetic expression
- Call Nested Procedures

Bracket Matching Problem

Ensures that pairs of brackets are properly matched

- Example: `{a,(b+f[4])*3,d+f[5]}`

- **Bad Examples:**

```
(..)..)    // too many closing brackets

(..(..)    // too many open brackets

[..(..]..)   // mismatched brackets
```

Push Open Bracket in stack -> { , [, (

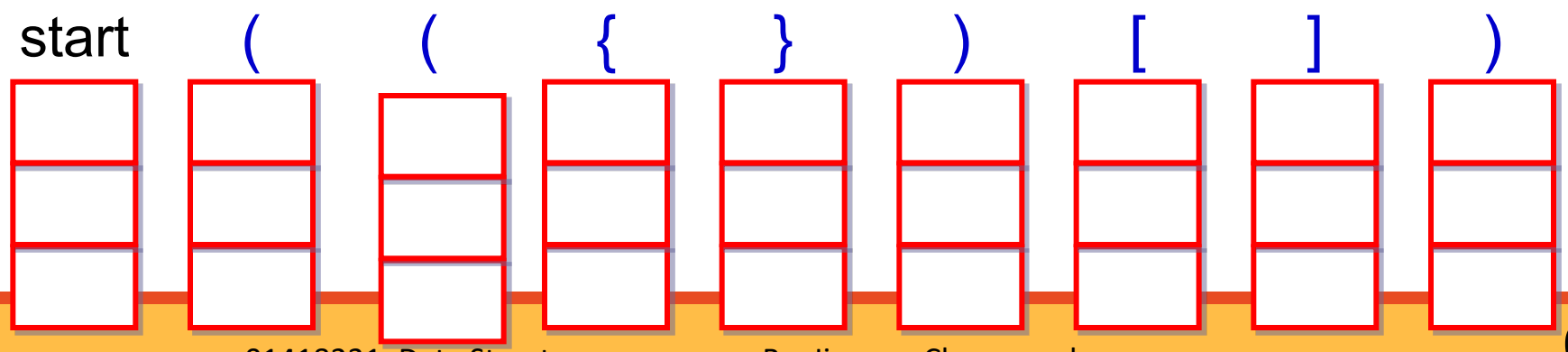
If operand is Close Bracket -> },],) check on top of stack

- if (it is the same type) then Pop stack
- else print “error”

Print result

- if (stack null) then print “OK”
- else print data in stack

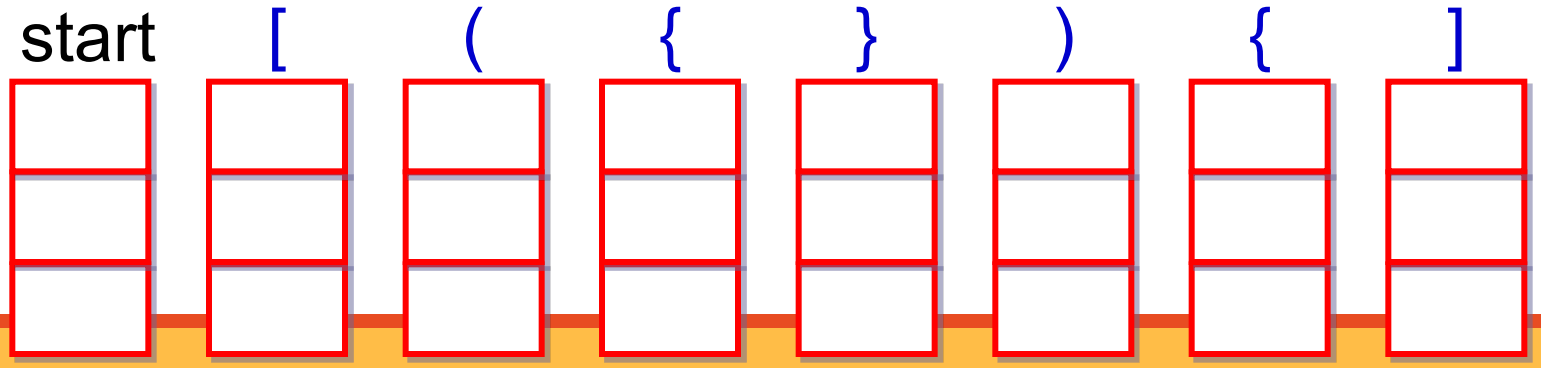
input : (({})[])



1. Push Open Bracket in stack -> { , [, (
2. If operand is Close Bracket -> },],) check on top of stack
   - if it is the same type then Pop stack
   - else print “error”

   Print result
   - if stack null then print “OK”
   - else print data in stack

input : [({}){])



Agenda

Application areas use stacks:

- Bracket Checker
- Convert Infix to Postfix
- Evaluation of arithmetic expression
- Call Nested Procedures

Arithmetic expression

Infix

- operand1 operator operand2

  1 + 2

Prefix

- operator operand1 operand2

  + 1 2

Postfix

- operand1 operand2 operator

  1 2 +

Arithmetic expression

Infix	Prefix	Postfix
A+B	+AB	AB+
(A+B)*C	*+ABC	AB+C*
A+(B*C)		
(A+B)/(C*D)		

Infix -> Postfix

อ่านนิพจน์ infix เข้าสู่โปรแกรมทีละ 1 ตัวอักษร

1) ถ้าข้อมูลที่อ่านเข้ามาเป็น operand ให้นำไปเป็น output

2) ถ้าข้อมูลที่อ่านเข้ามาเป็น (ให้ push (ลง stack

3) ถ้าข้อมูลที่อ่านเข้ามาเป็น) ให้ pop ข้อมูลออกจาก stack ไปเป็นผลลัพธ์ จนกว่าข้อมูลที่ pop ออกมาเป็น (ตัดวงเล็บปิด,เปิด ออกไป

Infix -> Postfix

4) **ถ้าข้อมูลที่อ่านเข้ามาเป็น operator** ให้ตรวจสอบว่า

- **ถ้า stack ว่าง** ให้ทำการ push operator ตัวนั้นลง stack
- **ถ้า stack ไม่ว่าง**
  1. นำไปเปรียบเทียบกับ operator ที่ top of stack
  2. ถ้าที่อ่านเข้ามามี priority น้อยกว่าหรือเท่ากันกับ top
     - pop operator ใน stack ไปที่ผลลัพธ์
  3. ถ้าที่อ่านเข้ามามี priority มากกว่า top หรือเจอ (
     - push operator ที่อ่านเข้ามาลง stack

5) **ถ้าหมดข้อมูล** ให้ pop สิ่งที่เหลือในสแตกออกไปที่ผลลัพธ์

Infix to Postfix expression A+B*C

Input (infix)	operator stack	output (postfix)

(A+B)*C

Input (infix)	operator stack	output (postfix)

INFIX : A + B * (C – D / E) / F

Input (infix)	operator stack	output (postfix)

INFIX : A + B * (C – D / E) / F

Input (prefix)	operator stack	output (postfix)

Agenda

Application areas use stacks:

- **Bracket Checker**
- **Convert Infix to Postfix**
- **Evaluation of arithmetic expression**
  - 2+3*4
- **Call Nested Procedures**

Evaluatio

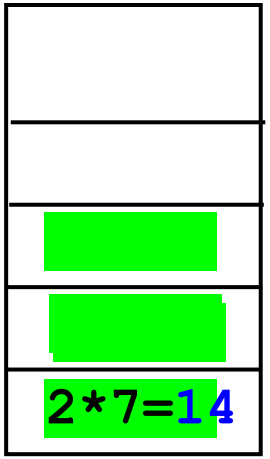
Initialize stack
For each item read.
If (it is an operand) then
push on the stack
If it is an operator then
pop arguments from stack;
perform operation;
push result onto the stack

Expr

Expr	
2	`s.push(2)`
3	`s.push(3)`
4	`s.push(4)`

+	`arg2=s.topAndPop()`
	`arg1=s.topAndPop()`
	`s.push(arg1+arg2)`

*	`arg2=s.topAndPop()`
	`arg1=s.topAndPop()`
	`s.push(arg1*arg2)`



Stack

Evaluation of arithmetic expression

- 623+-4+

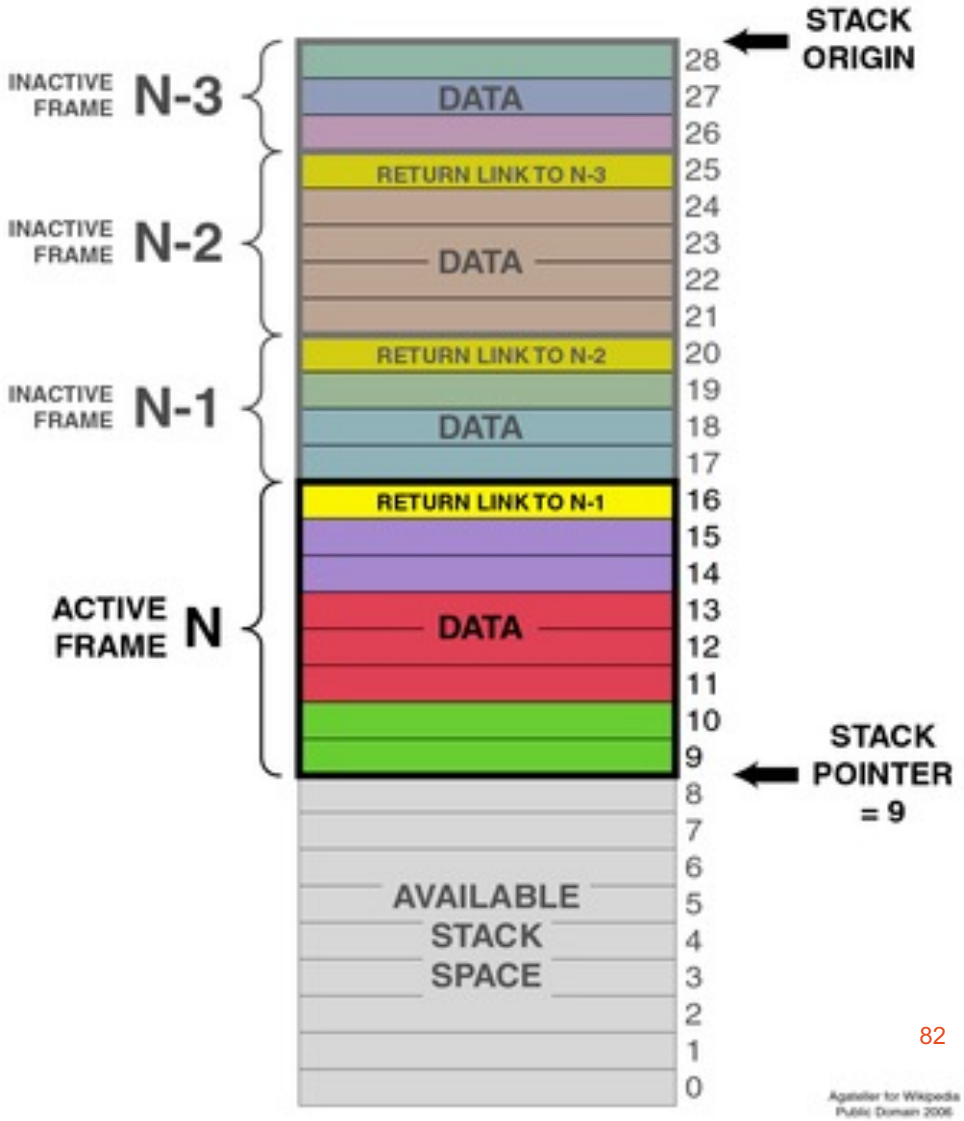
Input (postfix)	Operand 1	Operand 2	value	Operand stack

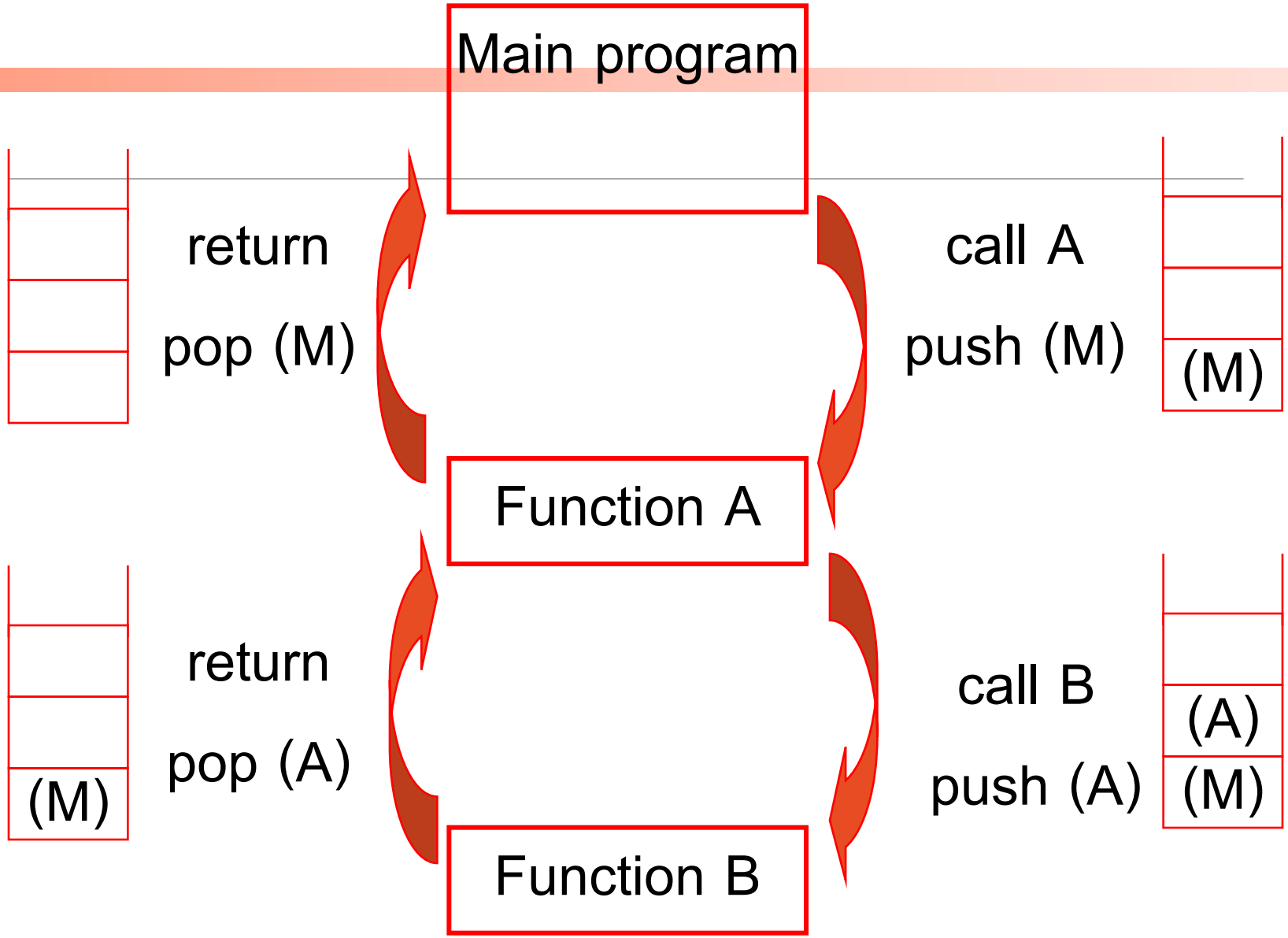
Agenda

Application areas use stacks:

- Bracket Checker
- Convert Infix to Postfix
- Evaluation of arithmetic expression
- Call Nested Procedures

Call Nested Procedures




Main program
return
pop (M)
call A
push (M)
(M)
Function A
return
pop (A)
(M)
call B
push (A)
(A)
(M)
Function B

Summary

Stack has many applications

- Bracket Checker
- Convert Infix to Postfix
- Evaluation of arithmetic expression
- Call Nested Procedures

Question