



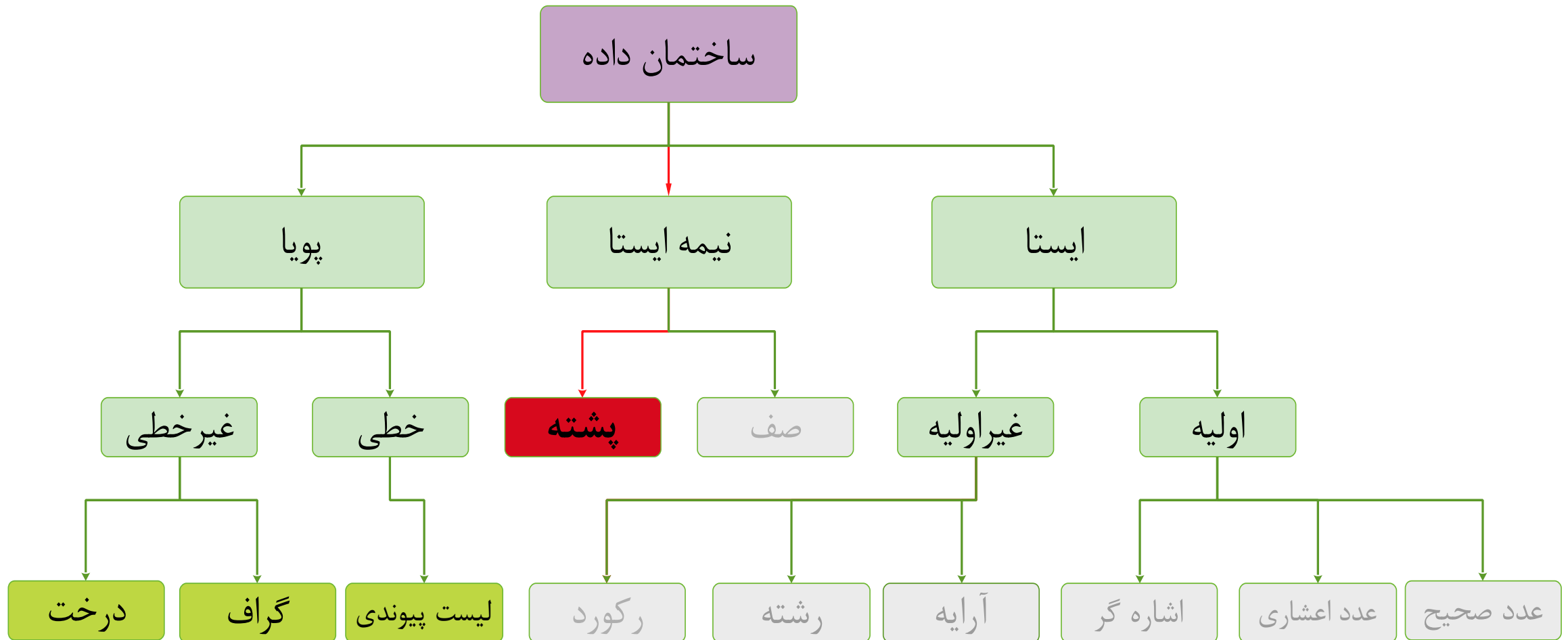
ساختمان داده‌ها و الگوریتم‌ها

فصل سوم

فهرست مطالب

- ❖ مقدمه‌ای بر الگوریتم‌ها و مفاهیم پایه
- ❖ معرفی پیچیدگی زمانی و حافظه‌ای و روشهای تحلیل مسائل
- ❖ **معرفی ساختمان داده‌های مقدماتی و الگوریتم‌های وابسته به آنها**
 - آرایه
 - صف
 - **پشته**
 - لیست پیوندی
- ❖ تئوری درخت و گراف و الگوریتم‌های مرتبط
- ❖ الگوریتم‌های مرتب‌سازی و تحلیل پیچیدگی مربوط به آنها
- ❖ مباحث تکمیلی در ساختمان داده‌ها

دسته بندی ساختمان داده‌ها



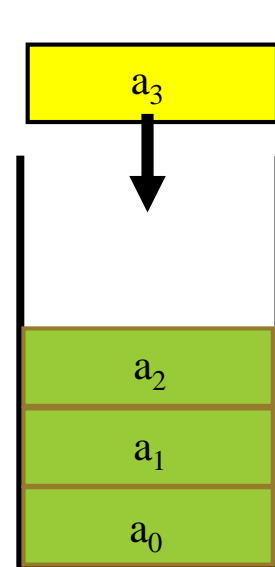
Stack (پشته)

What is a stack?

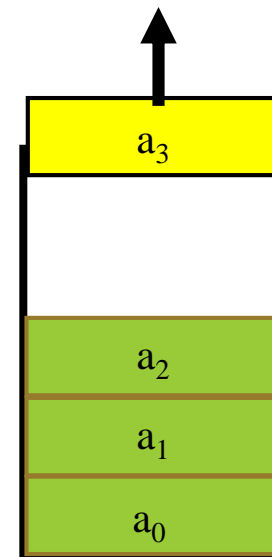
A stack is an ordered list in which insertions and deletions are made at one end called the top. It is also called a Last-In-First-Out (LIFO) list.

Stack (Cont.)

- Given a stack $S = (a_0, \dots, a_{n-1})$, a_0 is the bottom element, a_{n-1} is the top element, and a_i is on top of element a_{i-1} , $0 < i < n$.



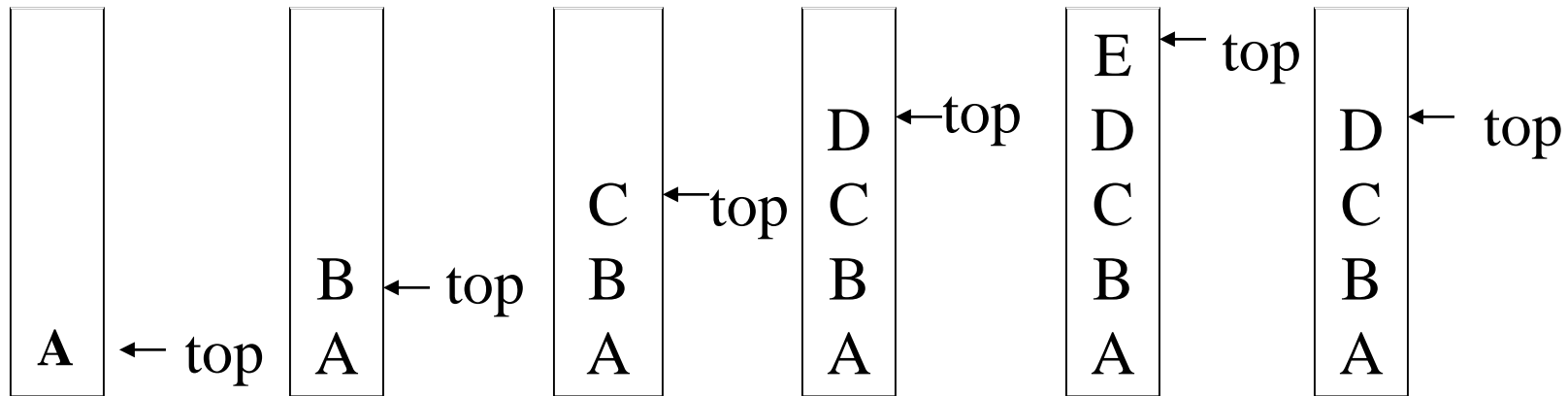
Push (Add)



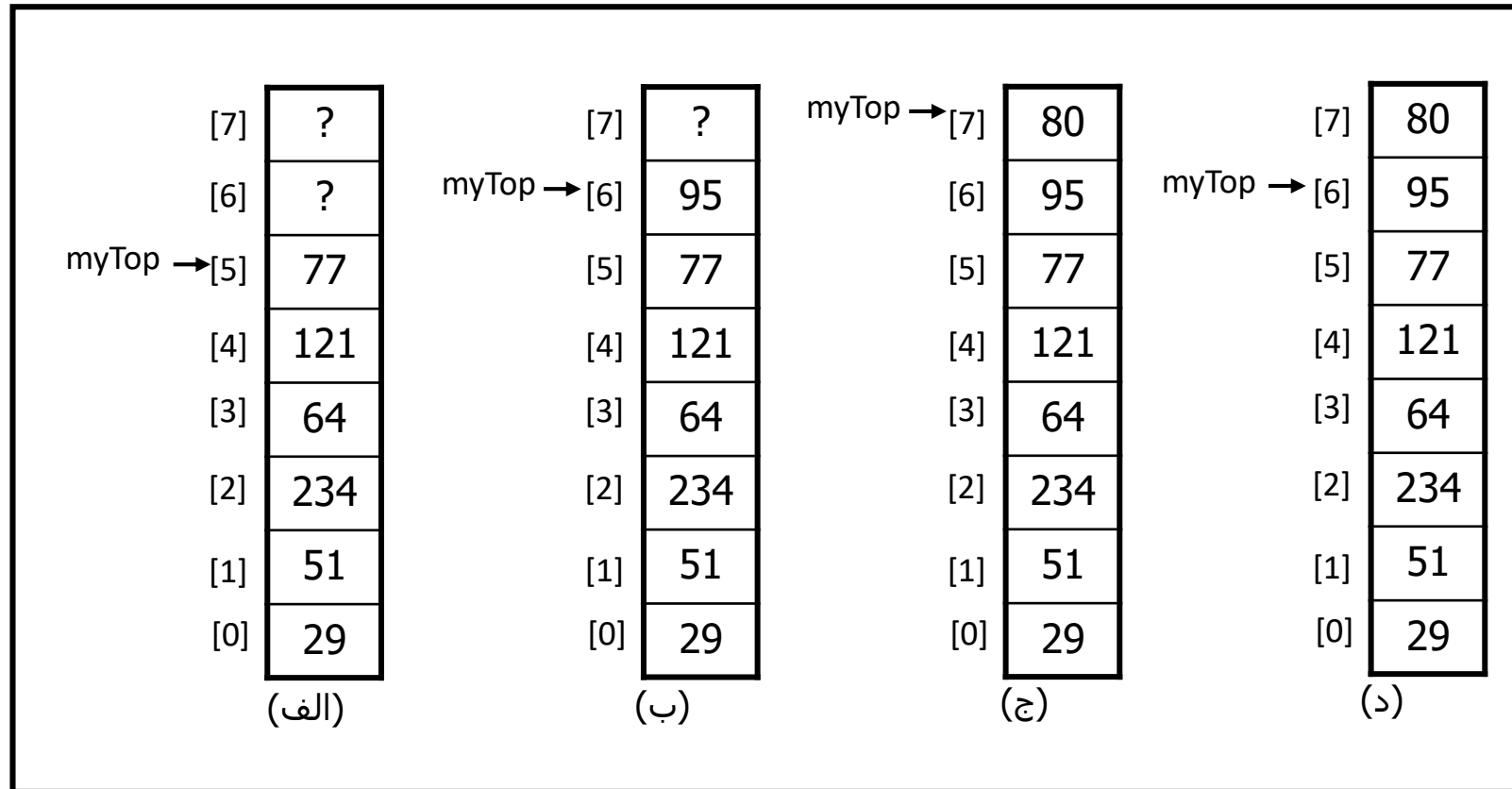
Pop (Delete)

Stack Example

S.Najjar.G@Gmail.com



Stack Example (Cont.)



Abstract Data Type for stack

structure *Stack* is

objects: a finite ordered list with zero or more elements.

functions:

for all $stack \in Stack$, $item \in element$, max_stack_size
 \in positive integer

Stack CreateS(max_stack_size) ::=

create an empty stack whose maximum size is
 max_stack_size

Boolean IsFull($stack$, max_stack_size) ::=

if (number of elements in $stack == max_stack_size$)

return TRUE

else return FALSE

Stack Add($stack$, $item$) ::=

if (IsFull($stack$)) $stack_full$

else insert $item$ into top of $stack$ and **return**

Abstract Data Type for stack (Cont.)

Boolean IsEmpty(*stack*) ::=

if(*stack* == CreateS(*max_stack_size*))

return TRUE

else return FALSE

Element Delete(*stack*) ::=

if(IsEmpty(*stack*)) **return**

else remove and return the *item* on the top
of the stack.

Implementation: using array

```
Stack CreateS(max_stack_size) ::=  
#define MAX_STACK_SIZE 100 /* maximum stack size */  
typedef struct {  
    int key;  
    /* other fields */  
} element;  
element stack[MAX_STACK_SIZE];  
int top = -1;
```

```
Boolean IsEmpty(Stack) ::= top < 0;
```

```
Boolean IsFull(Stack) ::= top >= MAX_STACK_SIZE-1;
```

Implementation: using array (Cont.)

Add to a stack

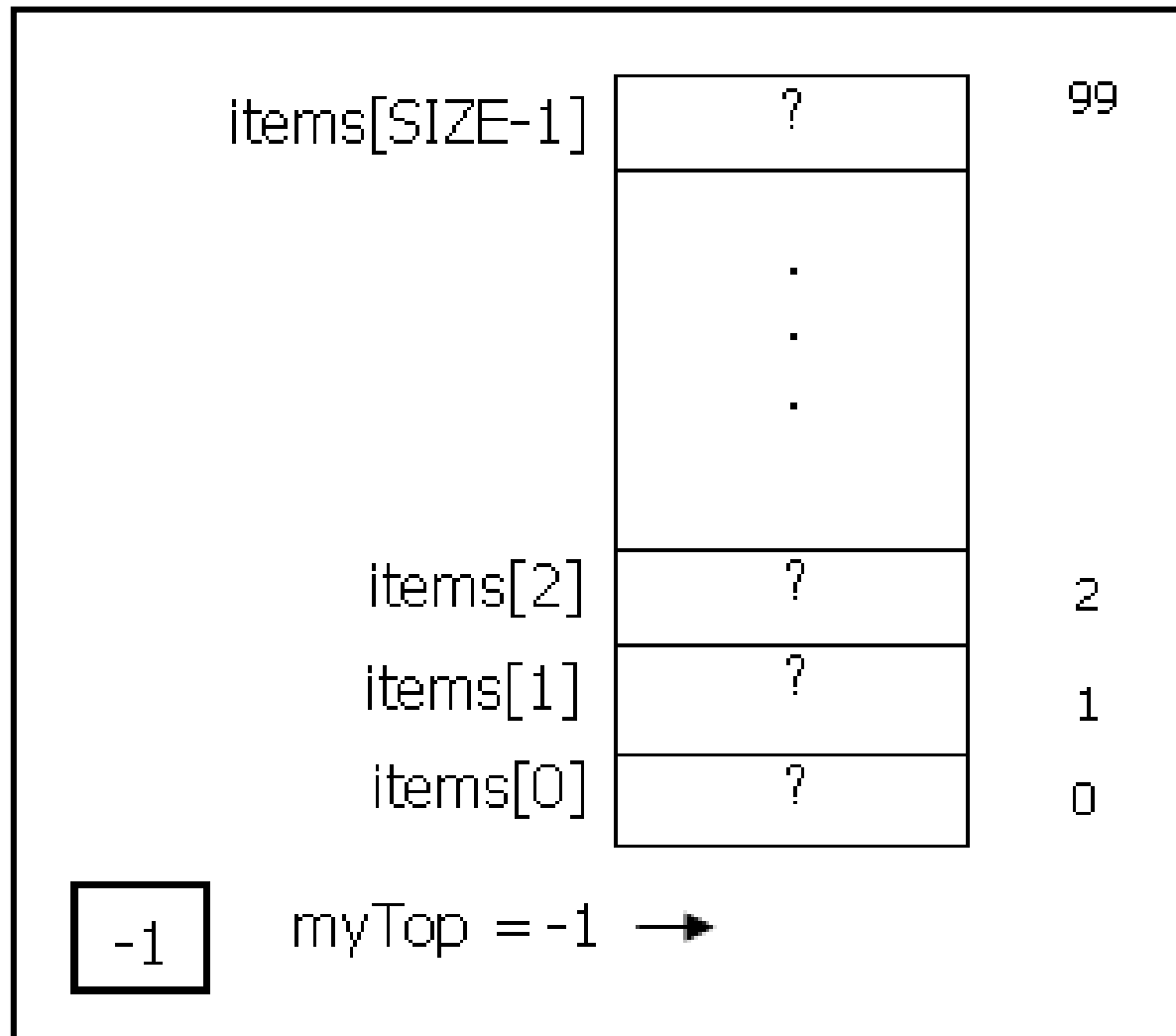
```
void Add(int top, element item)
{
    /* add an item to the global stack */
    if (top >= MAX_STACK_SIZE-1) {
        stack_full( );
        return;
    }
    stack[++top] = item;
}
```

Implementation: using array (Cont.)

Delete from a stack

```
element delete(int top)
{
    /* return the top element from the stack */
    if (top == -1)
        return stack_empty( ); /* returns and error key */
    return stack[top--];
}
```

Implementation: using array (Cont.)





Applications of Stack

- Expression evaluation and syntax parsing (represented in prefix, postfix or infix notations and conversion from one to another)
- Backtracking (finding the correct path in a maze)
- Runtime memory management (Stack-based memory allocation and Stack machine)



Home Work 7

S.Najjar.G@Gmail.com

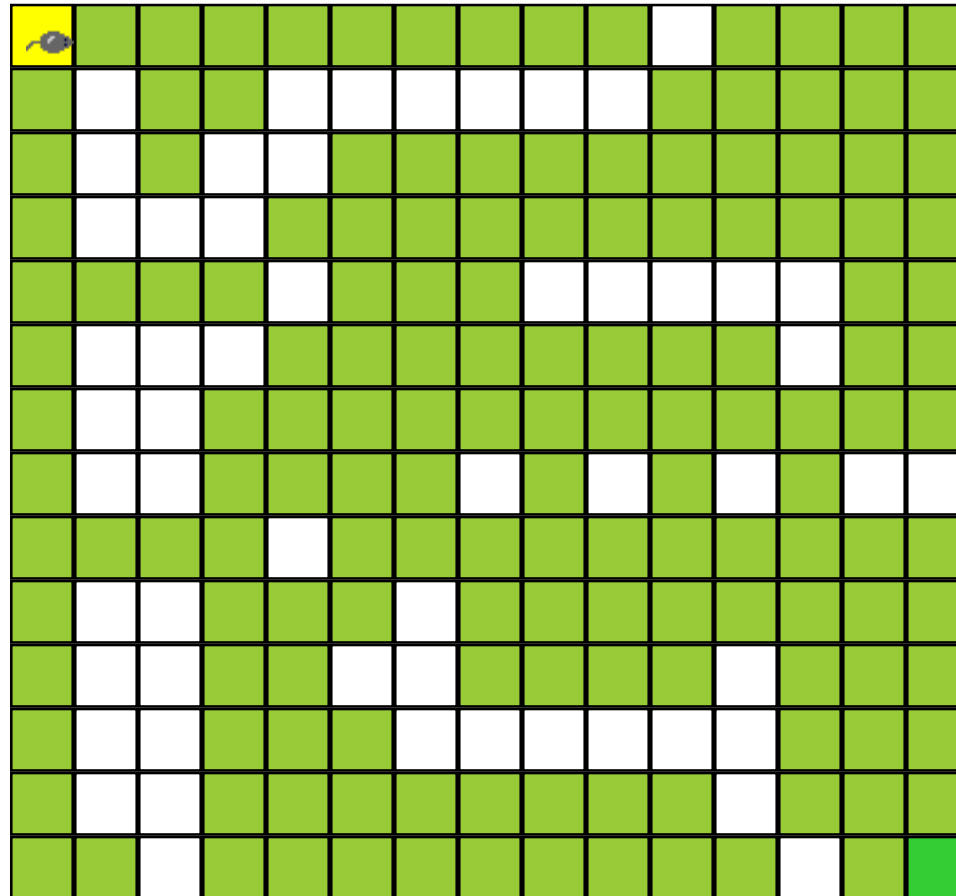
- الگوریتم **پشته** را با استفاده از آرایه در یکی از زبانهای برنامه نویسی پیاده سازی و مثالی روی آن اجراء نمایید.



کاربرد پشته در بازی پیچ و خم



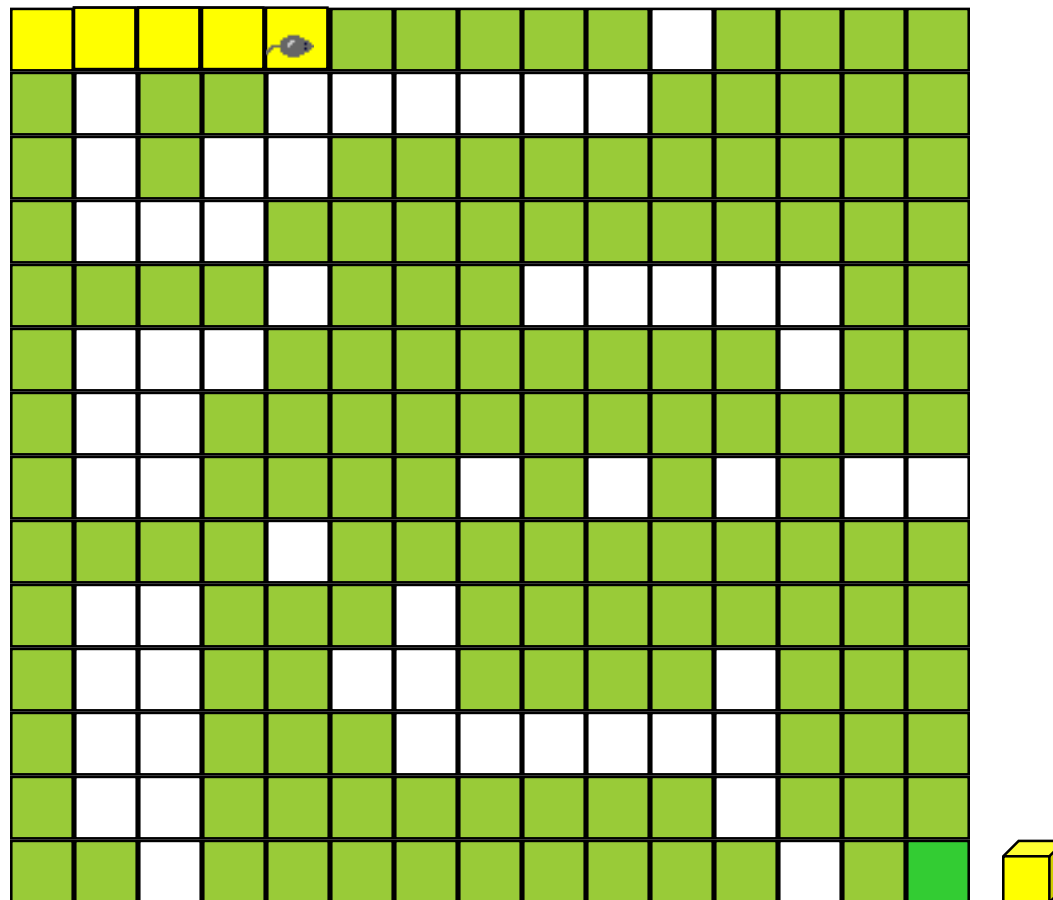
کاربرد پشته در بازی پیچ و خم



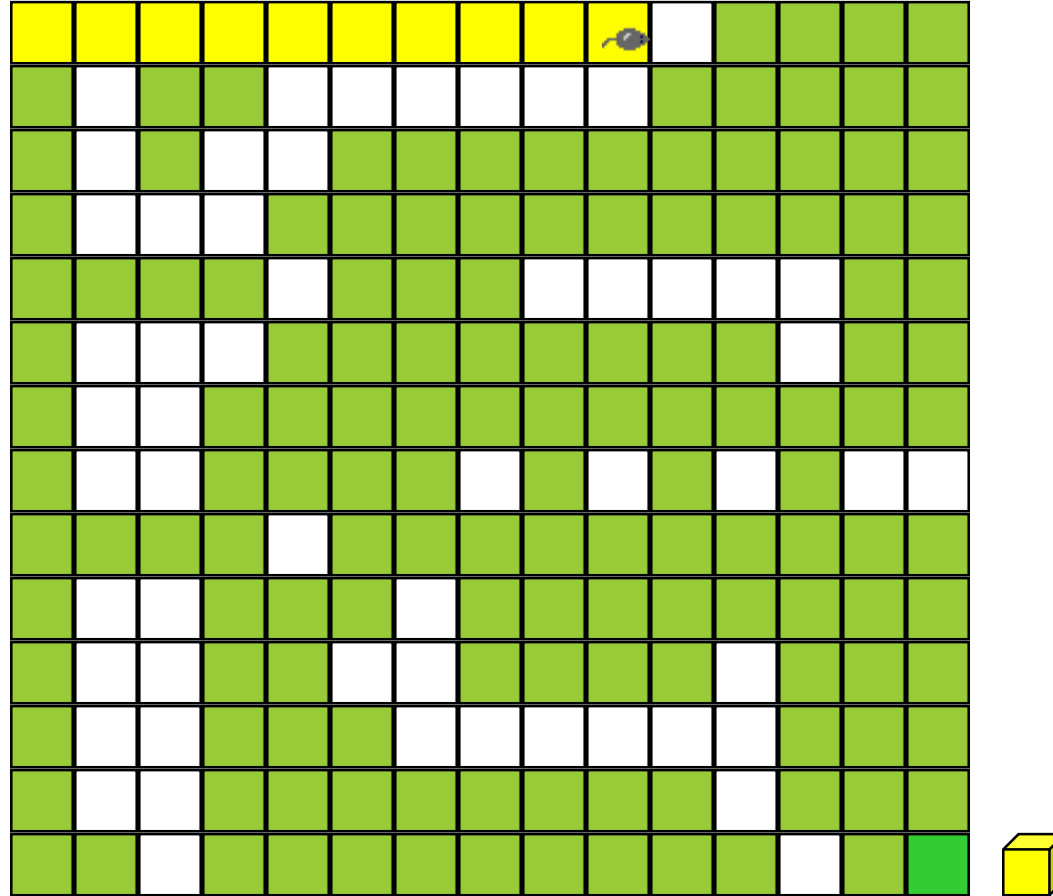
- حرکات مجاز: راست، پایین، چپ، بالا
- خانه ها را علامت گذاری کنید تا از بازدید مجدد جلوگیری شود.



کاربرد پشته در بازی پیچ و خم

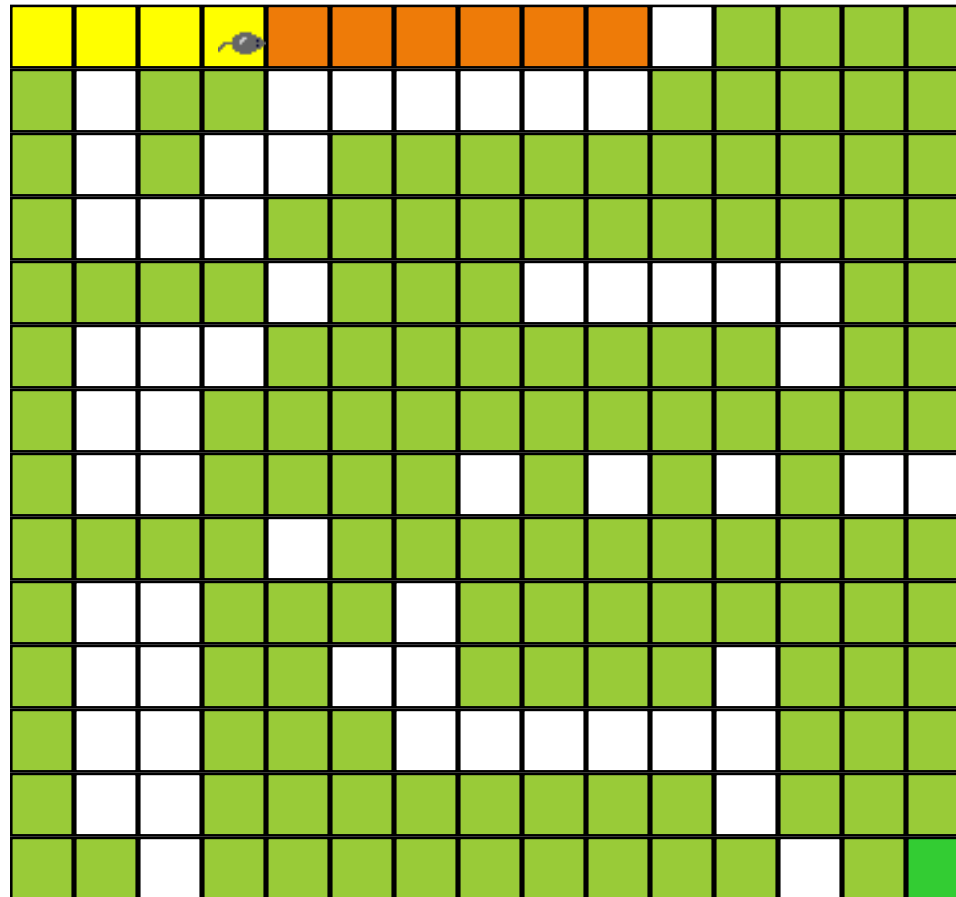


کاربرد پشته در بازی پیچ و خم



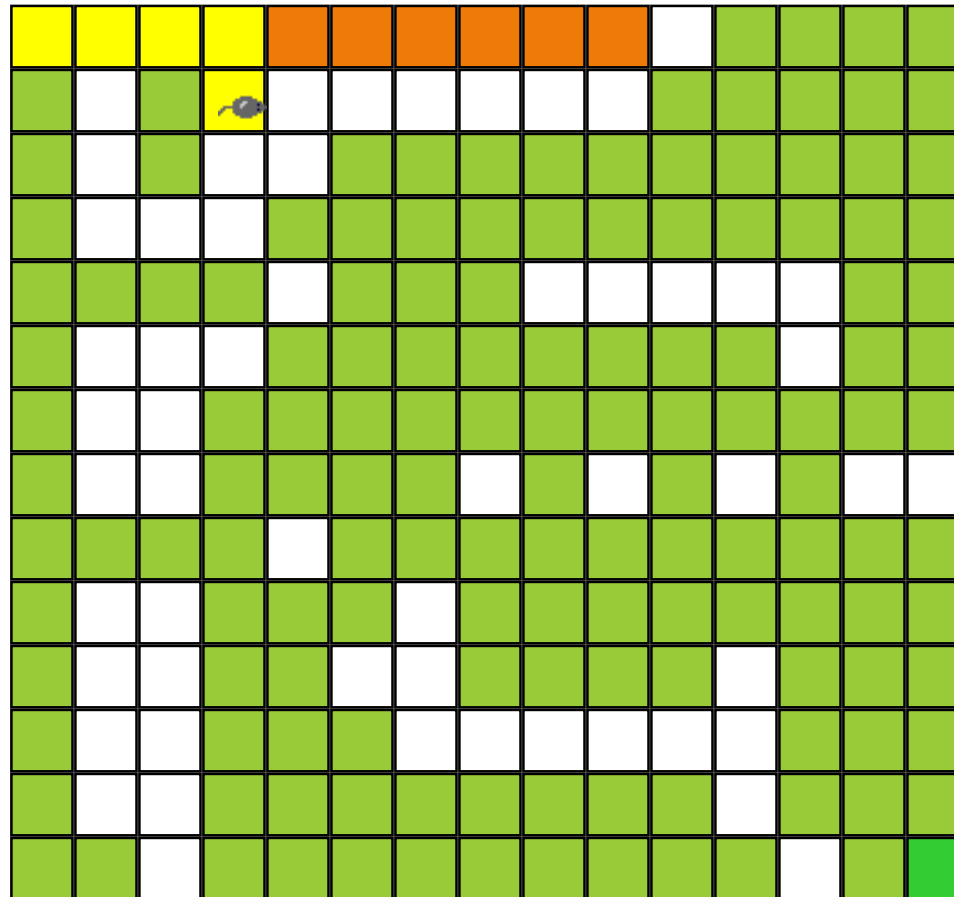
- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد

کاربرد پشته در بازی پیچ و خم



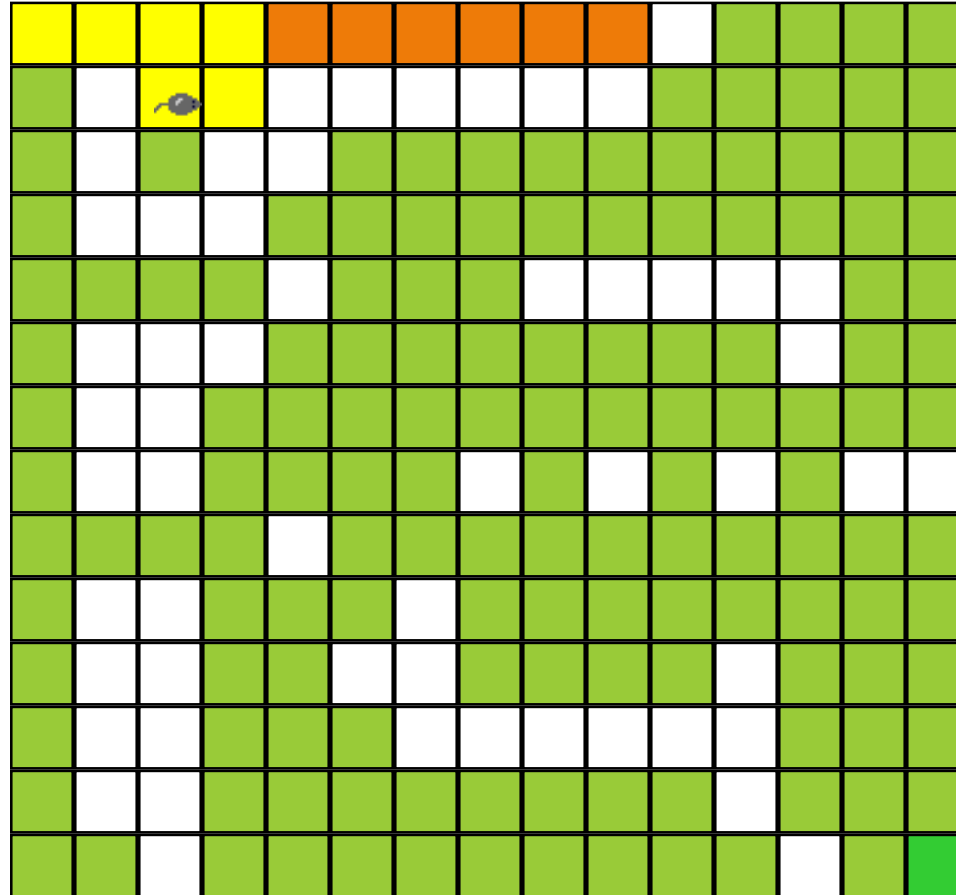
• به سمت پایین حرکت کنید.

کاربرد پشته در بازی پیچ و خم



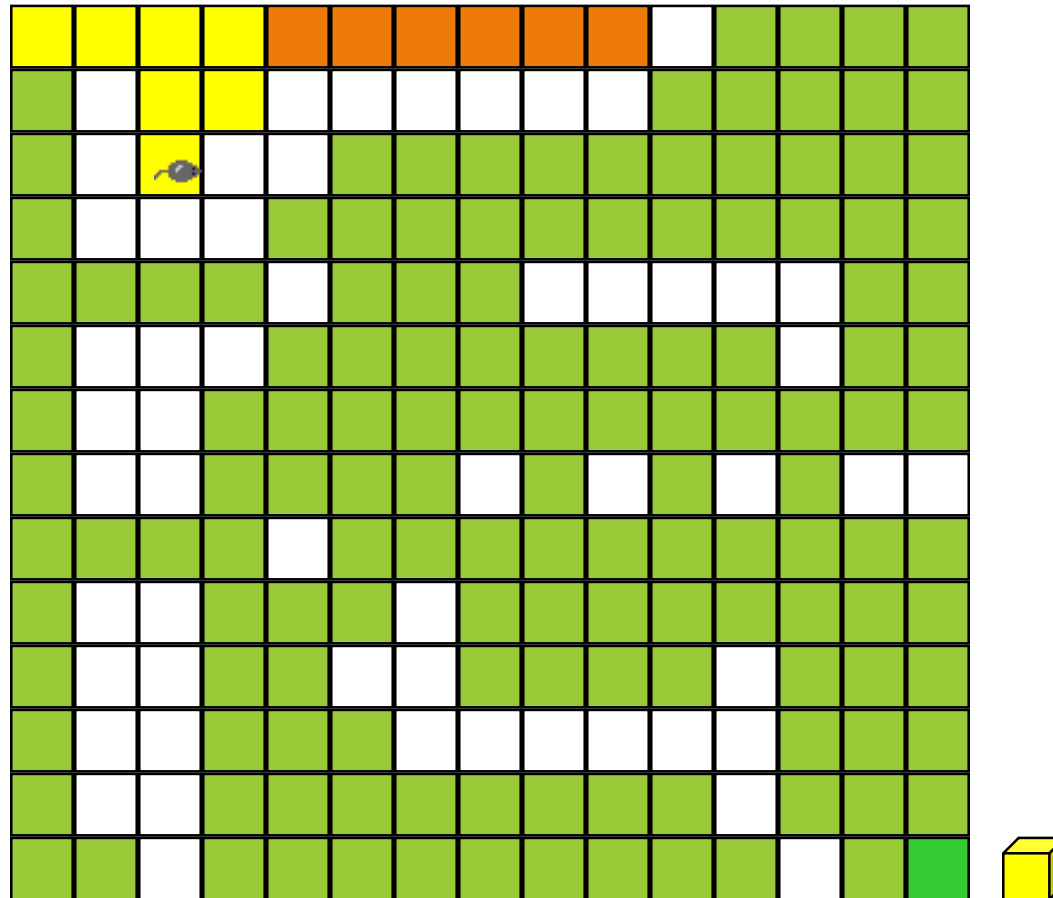
• به سمت چپ حرکت کنید.

کاربرد پشته در بازی پیچ و خم



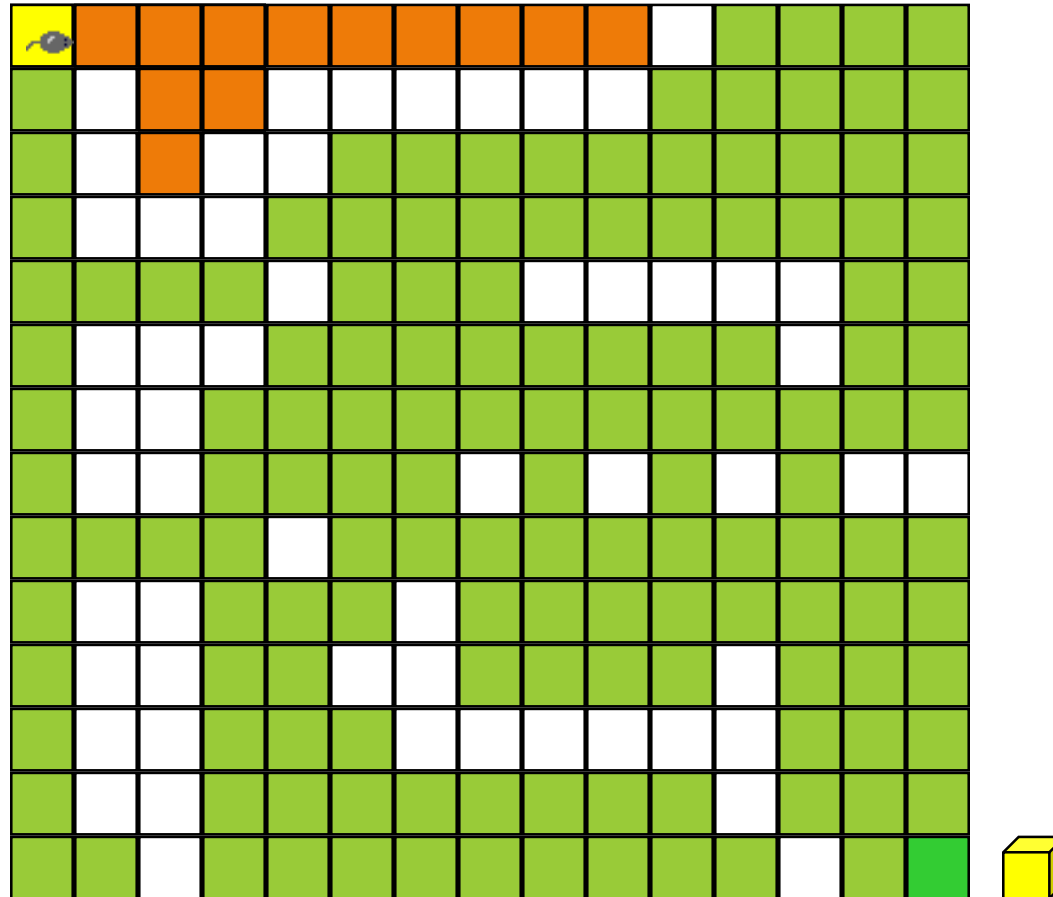
• به سمت پایین حرکت کنید.

کاربرد پشته در بازی پیچ و خم



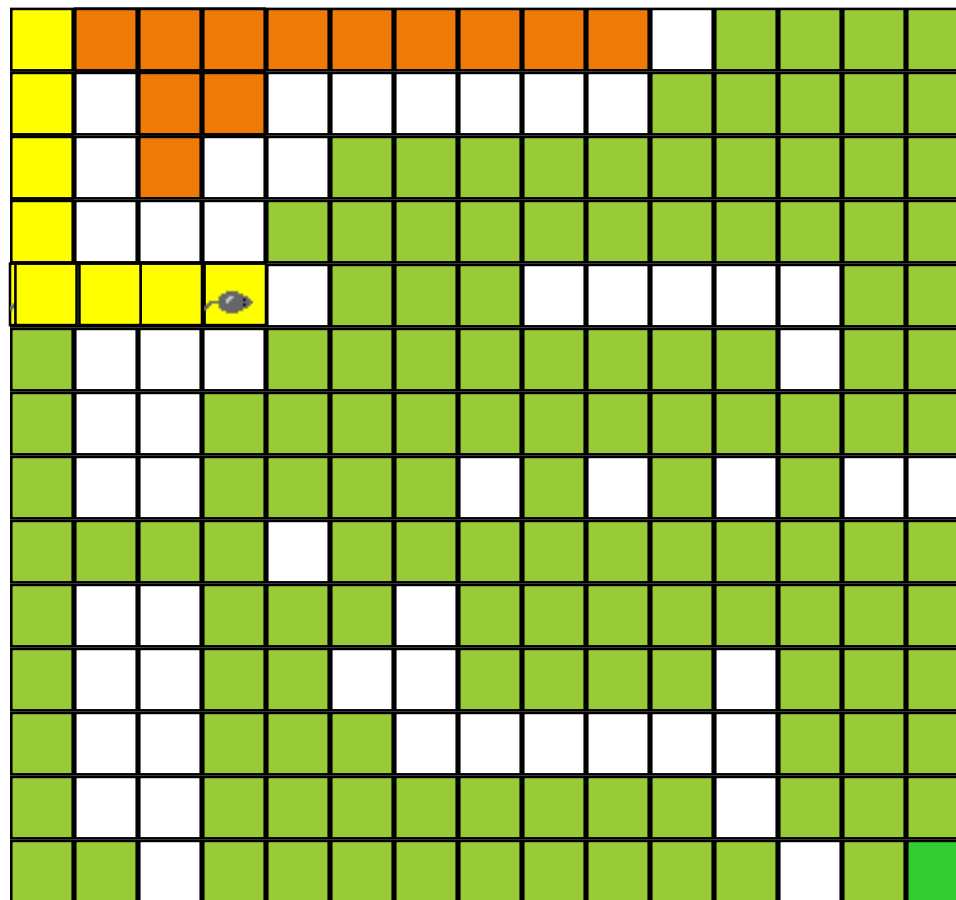
- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد

کاربرد پشته در بازی پیچ و خم

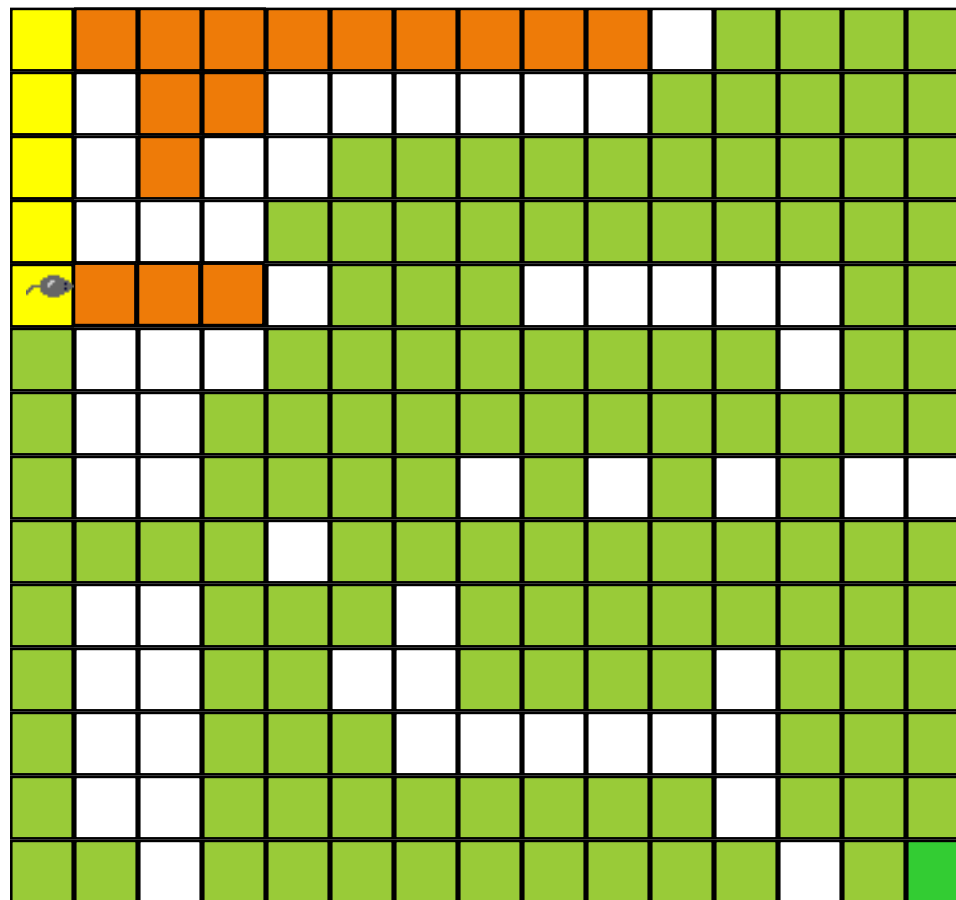


- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد.
- به سمت پایین حرکت کنید.

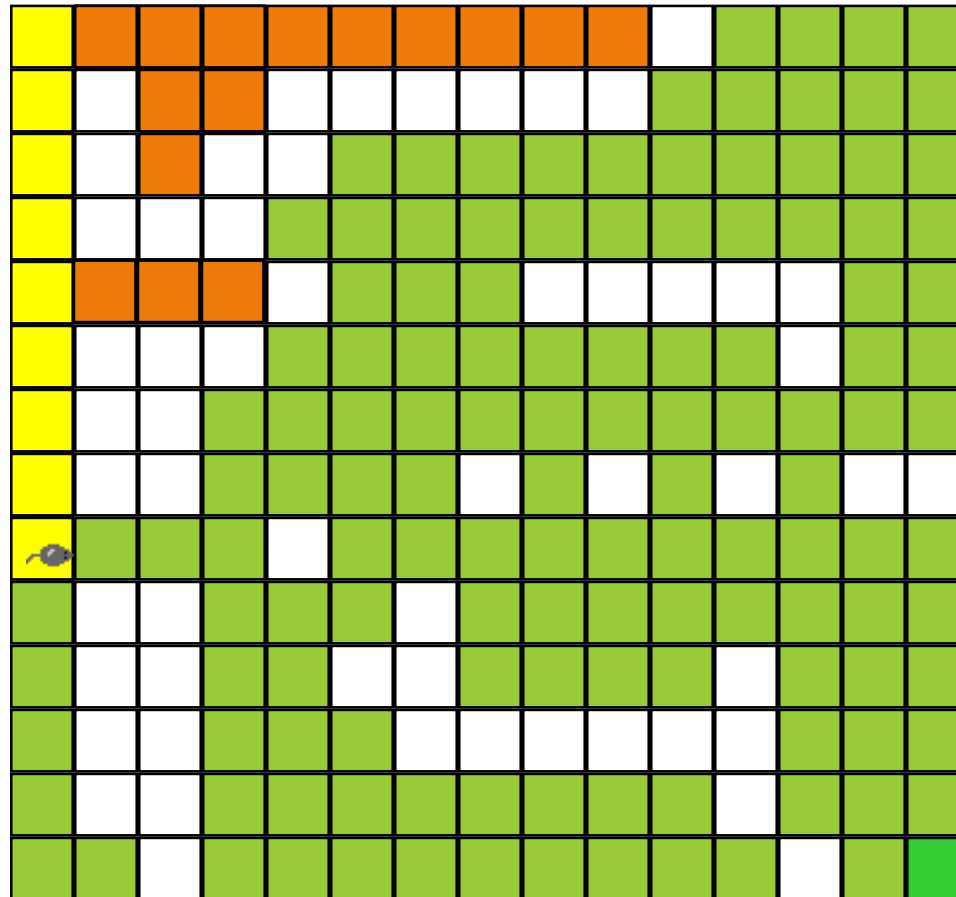
کاربرد پشته در بازی پیچ و خم



کاربرد پشته در بازی پیچ و خم

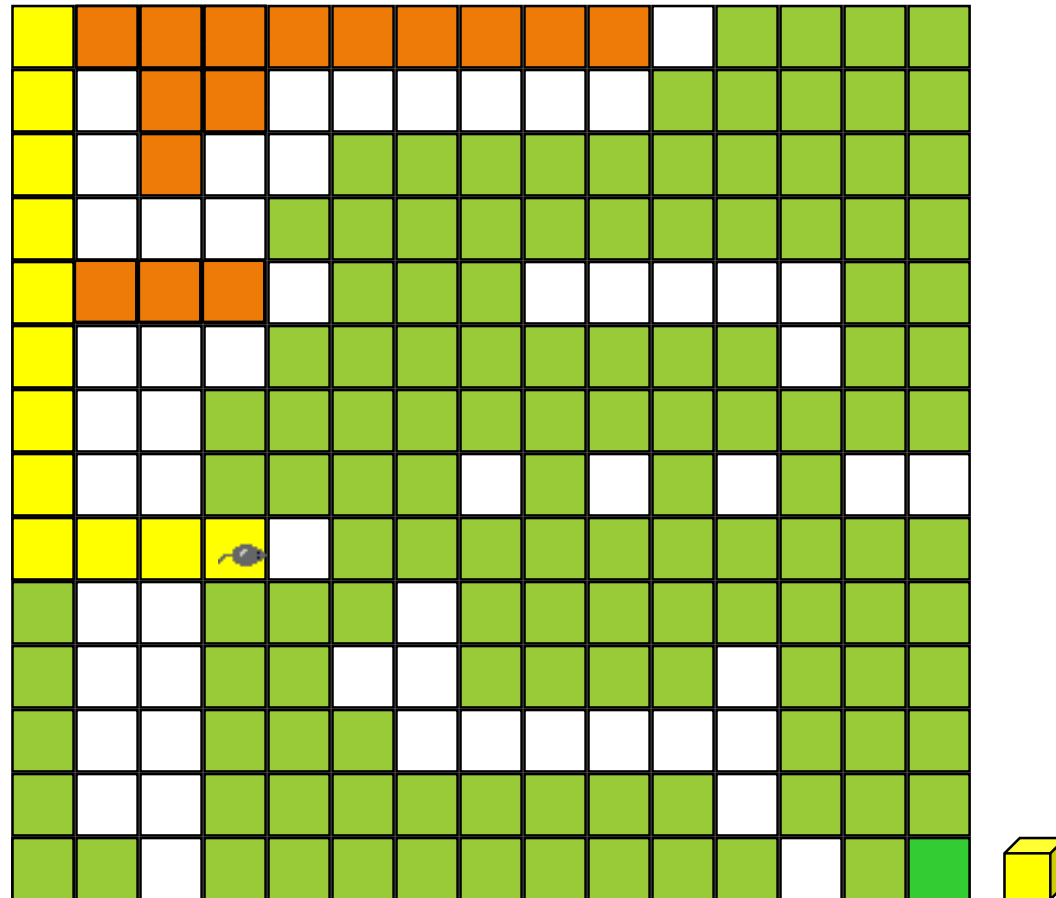


کاربرد پشته در بازی پیچ و خم



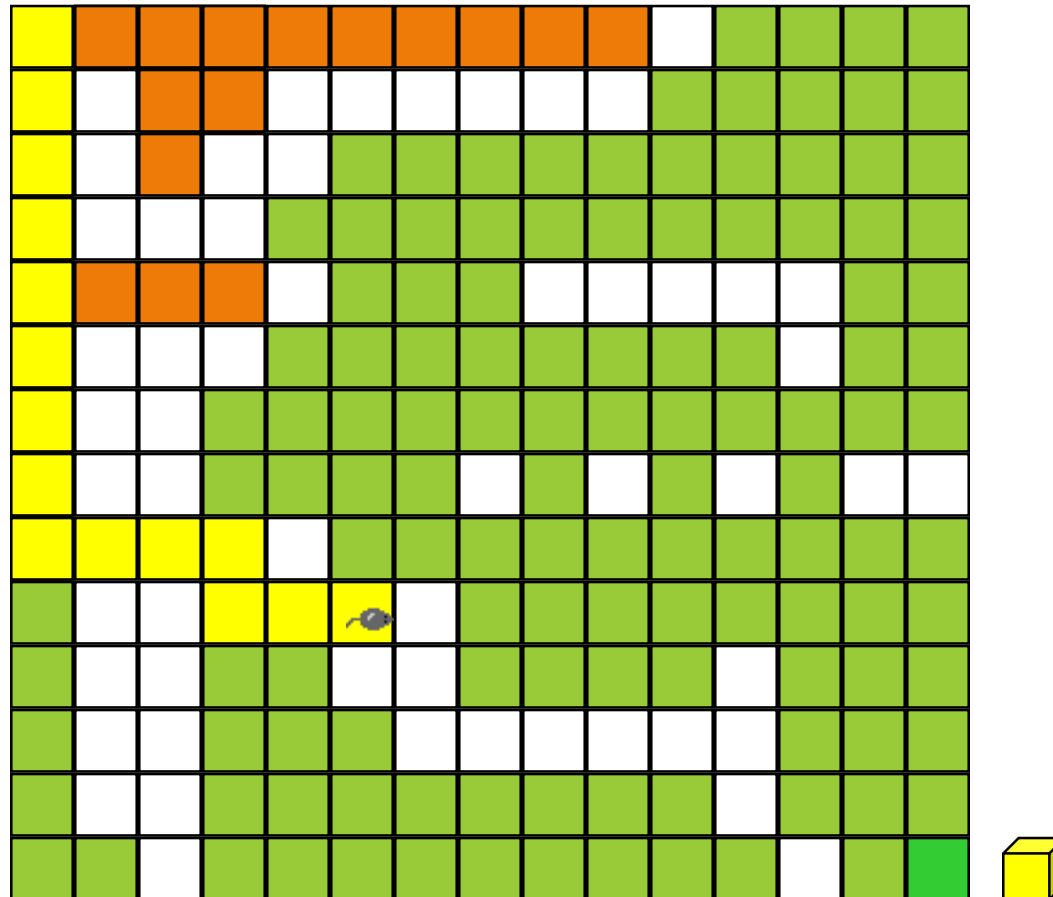
• به سمت راست حرکت کنید.

کاربرد پشته در بازی پیچ و خم



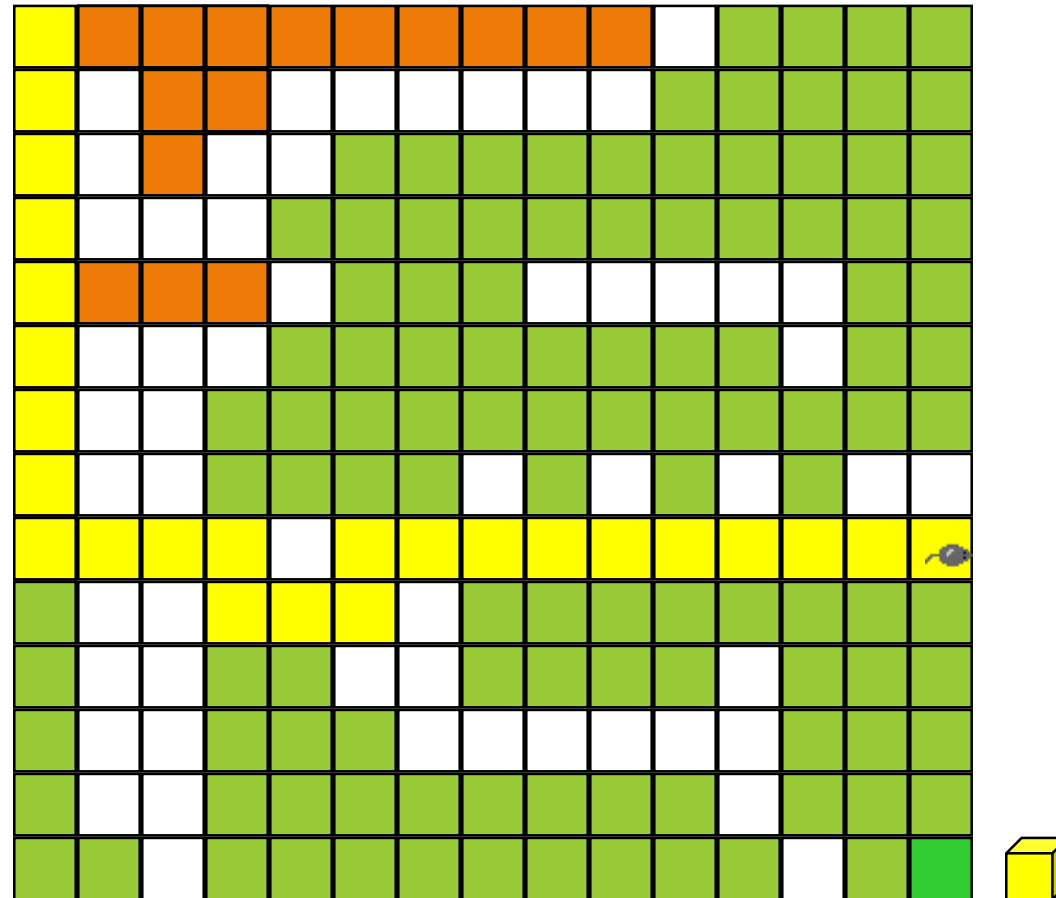
- یکی به سمت پایین و سپس به راست حرکت کنید.

کاربرد پشته در بازی پیچ و خم



- یکی به سمت بالا و سپس به راست حرکت کنید.

کاربرد پشته در بازی پیچ و خم



- به سمت پایین حرکت کرده و سپس از Maze خارج شوید.
- مسیر از شروع تا به پایان بر روی پشته قرار دارد.

ارزیابی عبارات

- $a = 4, b = c = 2, d = e = 3$
- value of $x = \underline{a/b - c + d * e - a * c}$
 - Interpretation 1: $((4/2)-2)+(3*3)-(4*2) = 0+8+9 = 1$
 - Interpretation 2: $(4/(2-2+3))*(3-4)*2 = (4/3)*(-1)*2 = -2.66666\dots$

□ با استفاده از پرانتزها می توان ترتیب اجرای عملیات را تغییر داد

- $x = ((a/(b - c+d))*(e - a)*c$

چگونه دستورات ماشین متناظر با یک کد را تولید کنیم؟

ارزیابی عبارات

Token	Operator	Precedence ¹	Associativity
() [] -> .	function call array element struct or union member	17	left-to-right
-- ++	increment, decrement ²	16	left-to-right
-- ++ ! ~ - + & * sizeof	decrement, increment ³ logical not one's complement unary minus or plus address or indirection size (in bytes)	15	right-to-left
(type)	type cast	14	right-to-left
* / %	multiplicative	13	left-to-right
+ -	binary add or subtract	12	left-to-right
<< >>	shift	11	left-to-right
> >= < <=	relational	10	left-to-right
== !=	equality	9	left-to-right
&	bitwise and	8	left-to-right
^	bitwise exclusive or	7	left-to-right
	bitwise or	6	left-to-right
&&	logical and	5	left-to-right
	logical or	4	left-to-right
?:	conditional	3	right-to-left
= += -= /= *= %= <<= >>= &= ^= =	assignment	2	right-to-left
,	comma	1	left-to-right

1. The precedence column is taken from Harbison and Steele.

2. Postfix form

3. Prefix form



ارزیابی عبارات

- InFix
- PostFix
- PreFix

ارزیابی عبارات

- کامپایلرها برای ارزیابی عبارات از بازنمایی میان ترتیب استفاده نمی کنند و به جای آن بازنمایی پس ترتیب را به کار می برند

Infix	Postfix
$2+3*4$	$2\ 3\ 4*+$
$a*b+5$	$ab*5+$
$(1+2)*7$	$1\ 2+7*$
$a*b/c$	$ab*c/$
$((a/(b-c+d))*(e-a))*c$	$abc-d+ /ea-*c*$
$a/b-c+d*e-a*c$	$ab/c-de*+ac*-$

ارزیابی عبارات

• تبدیل عبارت میانوندی به پسوندی یا پیشوندی

- با استفاده از پرانتزگذاری
- با استفاده از پشته
- با استفاده از پیمایش درخت محاسباتی

تبدیل عبارتهای میانوندی به پسوندی

رشته زیر را به فرم پسوندی تبدیل کنید

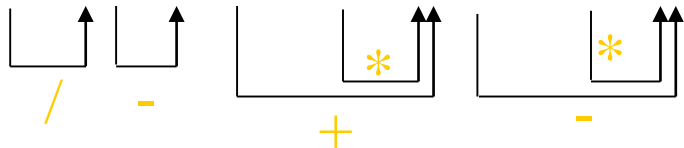
$$a / b - c + d * e - a * c$$

(۱) عبارت را به صورت کامل پرانتزگذاری کنید

$$a / b - c + d * e - a * c$$

$$((((a / b) - c) + (d * e)) - (a * c))$$

(۲) همه ی عملگرها جایگزین پرانتز راست متناظر خود می شوند.

$$((((a / b) - c) + (d * e)) - (a * c))$$


(۳) همه ی پرانتزها را حذف کنید.

$$a b / c - d e * + a c * -$$

ترتیب عملوندها در فرم میانوندی و پسوندی یکسان است

ارزیابی عبارات

• تبدیل عبارت میانوندی به پسوندی یا پیشوندی

- با استفاده از پرانتزگذاری
- با استفاده از پشته
- با استفاده از پیمایش درخت محاسباتی

ارزیابی عبارات

- تبدیل عبارت میانوندی به پسوندی:
 - رشته را از **چپ به راست** پیمایش کنید.
 - عملوندها مستقیماً در خروجی نوشته می شوند.
 - به هر «» پرانتز که رسیدیم آن را به راحتی در پشته قرار می دهیم.
 - به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای پشته **بیشتر** باشد، آن را در داخل پشته قرار می دهیم، در غیر این صورت آنقدر از بالای پشته عملگر خارج میکنیم که یا به عملگری برسیم که بتوانیم با توجه به اولویت آن را در پشته قرار دهیم یا اینکه به آخر پشته برسیم.
 - بر روی «» هر عملگری میتواند قرار بگیرد، تا موقعی که به «» برسیم در این حالت آنقدر از پشته عملگر حذف می کنیم که به «» برسیم. در این وضعیت این دو نماد همدیگر را خنثی می کنند.
 - زمانی که پیمایش عبارت تمام شد، تمام عناصر پشته را خالی می کنیم و در خروجی می نویسیم.

مثال تبدیل عبارت میانوندی به پسوندی با استفاده از پشته

Token	Stack			Top	Output
	[0]	[1]	[2]		
<i>a</i>				-1	<i>a</i>
+	+			0	<i>a</i>
<i>b</i>	+			0	<i>ab</i>
*	+	*		1	<i>ab</i>
<i>c</i>	+	*		1	<i>abc</i>
<i>eos</i>				-1	<i>abc*+</i>

$a+b*c$

$abc*+$

Token	Stack			Top	Output
	[0]	[1]	[2]		
<i>a</i>				-1	<i>a</i>
*	*			0	<i>a</i>
(*	(1	<i>a</i>
<i>b</i>	*	(1	<i>ab</i>
+	*	(+	2	<i>ab</i>
<i>c</i>	*	(+	2	<i>abc</i>
)	*			0	<i>abc +</i>
*	*			0	<i>abc +*</i>
<i>d</i>	*			0	<i>abc +*d</i>
<i>eos</i>	*			0	<i>abc +*d*</i>

$a*(b+c)*d$

$abc+*d*$

ارزیابی عبارات

- تبدیل عبارت میانوندی به پسوندی:
 - رشته را از **راست به چپ** پیمایش کنید.
 - عملوندها مستقیماً در خروجی نوشته می شوند.
 - به هر «)» پرانتز که رسیدیم آن را به راحتی در پشته قرار می دهیم.
 - به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای پشته **بیشتر یا برابر** باشد، آن را در داخل پشته قرار می دهیم، در غیر این صورت آنقدر از بالای پشته عملگر خارج می کنیم که یا به عملگری برسیم که بتوانیم با توجه به اولویت آن را در پشته قرار دهیم یا اینکه به آخر پشته برسیم.
 - بر روی «)» هر عملگری میتواند قرار بگیرد، تا موقعی که به «)» برسیم در این حالت آنقدر از پشته عملگر حذف می کنیم که به «)» برسیم. در این وضعیت این دو نماد همدیگر را خنثی می کنند.
 - زمانی که پیمایش عبارت تمام شد، تمام عناصر پشته را خالی می کنیم و در خروجی می نویسیم.

ارزیابی عبارات

• تبدیل عبارت میانوندی به پسوندی یا پیشوندی

- با استفاده از پرانتزگذاری
- با استفاده از پشته
- با استفاده از پیمایش درخت محاسباتی

ارزیابی عبارات

- تبدیل عبارت میانوندی به پسوندی یا پیشوندی

- با استفاده از پیمایش درخت محاسباتی

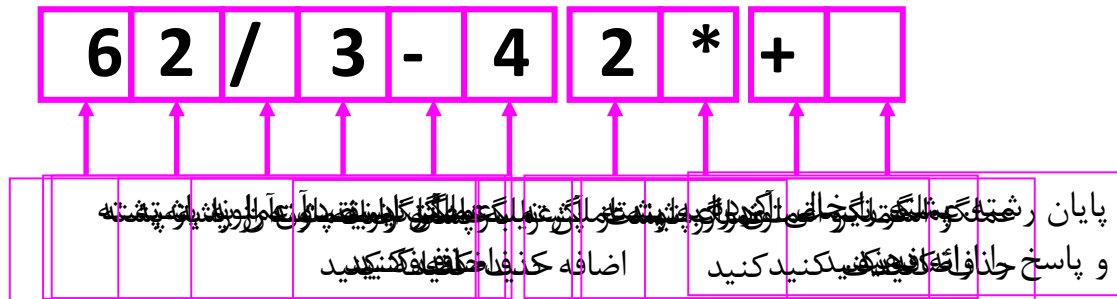


تبدیل پسوندی به میانوندی

فقط یک پیمایش از چپ به راست از روی رشته انجام می شود

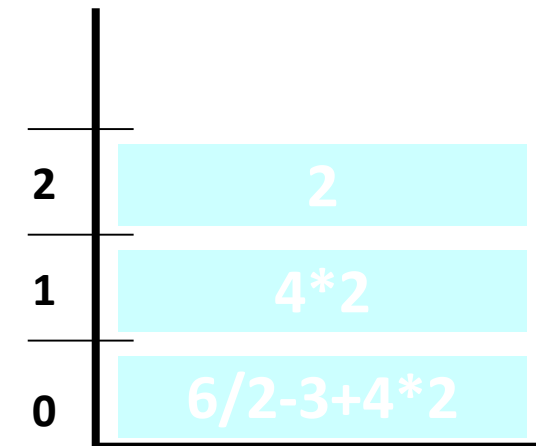
string: 6 2/3-4 2*+

حاصل عبارت را با عملگر مربوطه به دست آورده و در پشته قرار دهید



پاسخ عبارت است از $6 / 2 - 3 + 4 * 2$

top →



STACK

شود top-1 اکنون باید