

# ساختمان داده‌ها

فصل سوم

پشته‌ها و صفحه‌ها

E-mail: **Hadi.khademi@gmail.com**

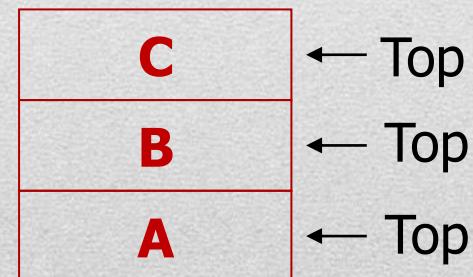
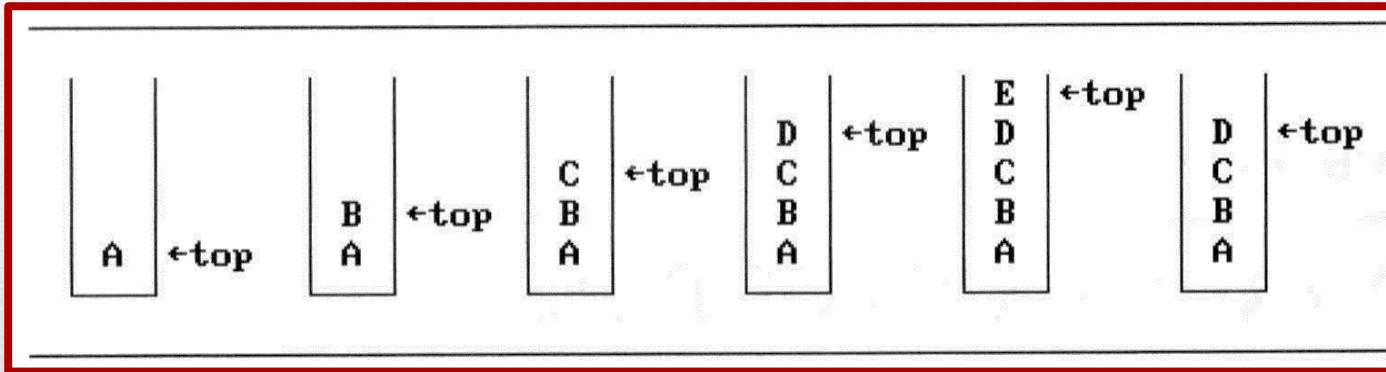
مهر ماه ۹۴ - دانشگاه علم و فرهنگ

---

- پشته ها به عنوان یک نوع داده مجرد
- صفات ها به عنوان یک نوع داده مجرد
- چند کاربرد از پشته
  - تطبیق پرانتزهای یک عبارت
    - بازی Maze
    - ارزیابی عبارات
    - تبدیل عبارتهای میانوندی به پسوندی

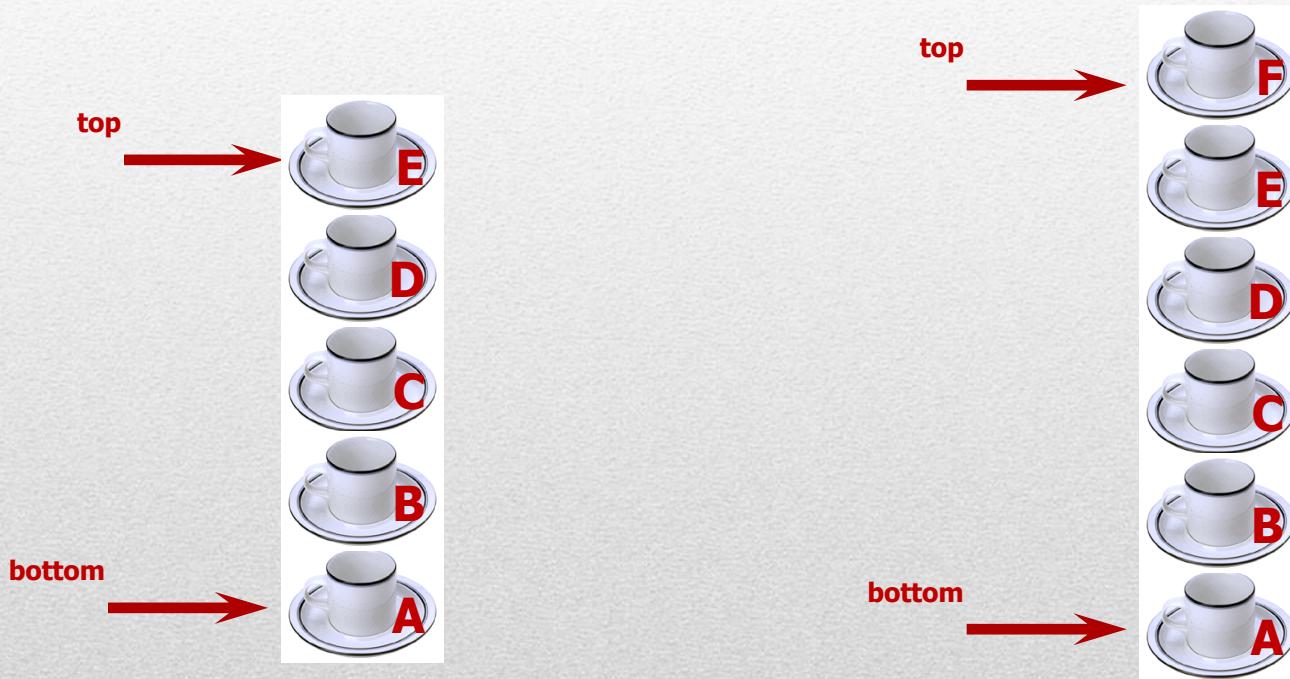
# پشته ها و صفات

- پشته ها حالت خاصی از لیست های مرتب شده می باشند که جایگذاری و حذف از یک سمت آن که top (بالا) نامیده می شود ، صورت می گیرد.
- پشته یک لیست (Last-In-First-Out) LIFO نیز نامیده می شود.



# پشته

- در پشته ای مانند  $a_0, S = a_0, \dots, a_{n-1}$  عنصر پایینی و عنصر بالایی می باشد.



# پشته

**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**

$Boolean IsEmpty(stack) ::=$

**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.

---

پسته ها به عنوان یک نوع داده مجرد

*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;*

# پسته ها به عنوان یک نوع داده مجرد

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;
}
```

item

←Top

stack

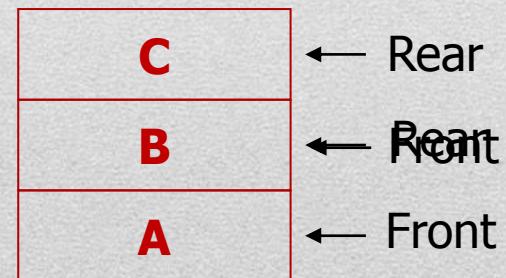
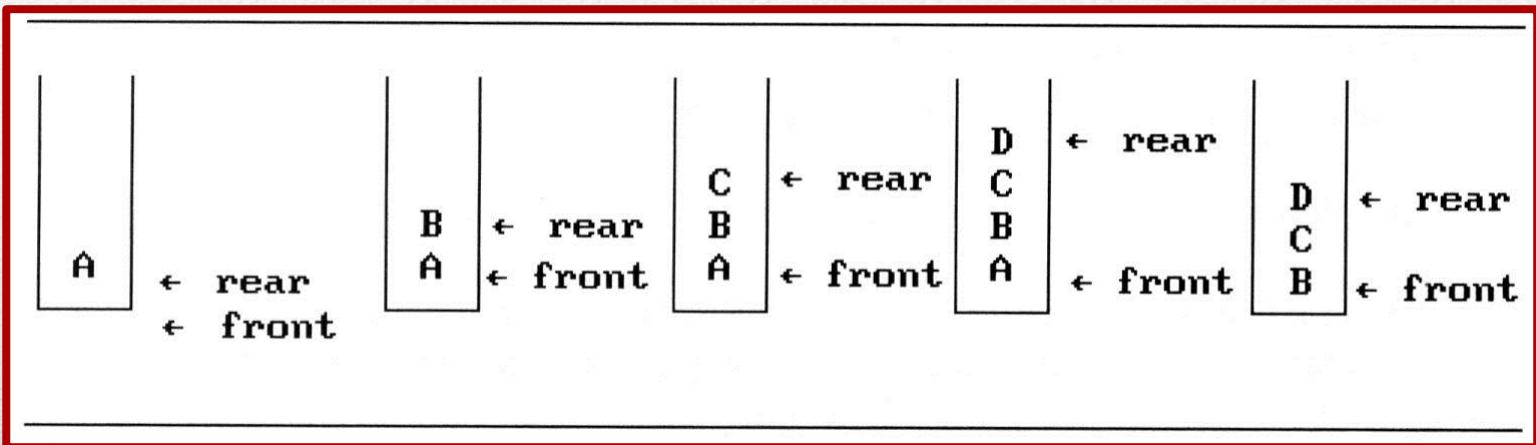
```
element delete(int *top)
{
/* return the top element from the stack */
if (*top == -1)
    return stack_empty(); /* returns an error key */
return stack[(*top)--];
}
```

item

←Top

# پشته ها به عنوان یک نوع داده مجرد

- صفها حالت خاصی از لیست های مرتب شده می باشند که جایگذاری از یک سمت آن که front (ابتدا) نامیده می شود و حذف از سمت دیگر که (انتها) نامیده می شود صورت می گیرد.
- صف یک لیست FIFO (First-In-First-Out) نیز نامیده می شود.



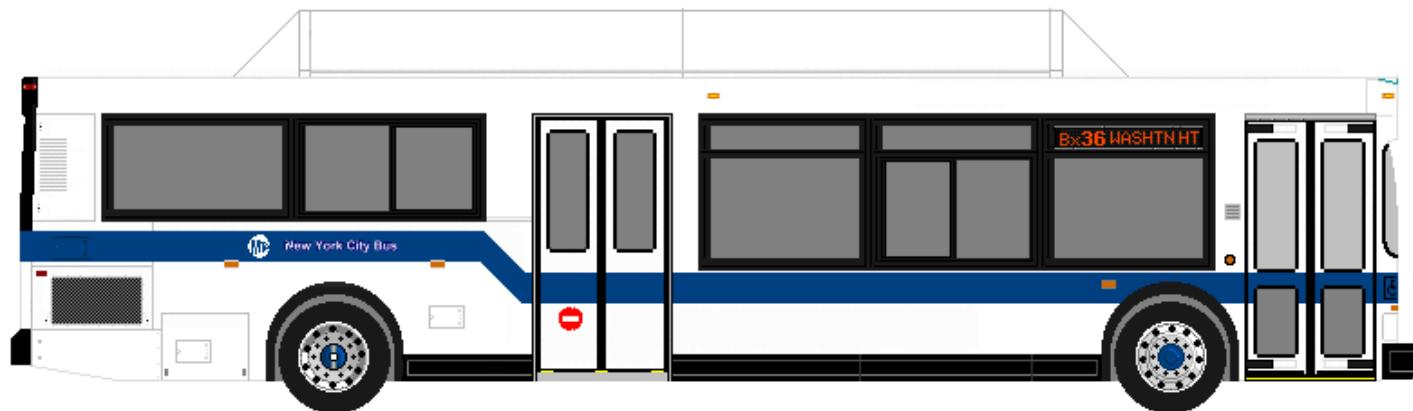
# صف



Bus  
Stop

front

rear



صف اتوبوس

E-mail: Hadi.khademi@gmail.com

**structure** *Queue* is

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

**functions:**

for all *queue* ∈ *Queue*, *item* ∈ *element*, *max-queue-size* ∈ positive integer

*Queue* CreateQ(*max-queue-size*) ::=

create an empty queue whose maximum size is *max-queue-size*

*Boolean* IsFullQ(*queue*, *max-queue-size*) ::=

**if** (number of elements in *queue* == *max-queue-size*)

**return** TRUE

**else return** FALSE

*Queue* AddQ(*queue*, *item*) ::=

**if** (IsFullQ(*queue*)) *queue - full*

**else** insert *item* at rear of *queue* and return *queue*

*Boolean* IsEmptyQ(*queue*) ::=

**if** (*queue* == CreateQ(*max-queue-size*))

**return** TRUE

**else return** FALSE

*Element* DeleteQ(*queue*) ::=

**if** (IsEmptyQ(*queue*)) **return**

**else** remove and return the *item* at front of *queue*.

صف ها به عنوان یک نوع داده مجرد

▪ پياده سازی: با کمک آرایه یک بعدی و دو متغیر rear و front

یک محل قبل از اولین عنصر	front
محل آخرین عنصر	rear

*Queue CreateQ(max-queue-size) ::=*

```
#define MAX_QUEUE_SIZE 100 /*Maximum queue size*/  
typedef struct {  
    int key;  
    /* other fields */  
    } element;  
element queue[MAX_QUEUE_SIZE];  
int rear = -1;  
int front = -1;
```

*Boolean IsEmptyQ(queue) ::= front == rear*

*Boolean IsFullQ(queue) ::= rear == MAX\_QUEUE\_SIZE-1*

## صف ها به عنوان یک نوع داده مجرد

```
void addq(int *rear, element item)
{
/* add an item to the queue */
if (*rear == MAX_QUEUE_SIZE-1) {
    queue_full();
    return;
}
queue[++*rear] = item;
}
```

```
element deleteq(int *front, int rear)
{
/* remove element at the front of the queue */
if (*front == rear)
    return queue_empty(); /*return an error key */
return queue[++*front];
}
```

# صف ها به عنوان یک نوع داده مجرد

- مشکل ممکن است با وجود آنکه در صف جای خالی وجود دارد  $\text{IsFullQ}$  برابر true شود.

مثال

$front$	$rear$	$Q[0]$	$Q[1]$	$Q[2]$	$Q[3]$	Comments
-1	-1					queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

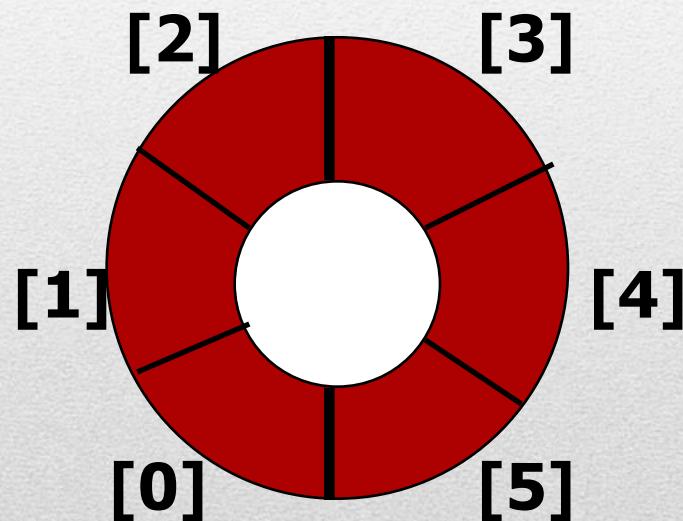
- صف به تدریج به سمت راست شیفت پیدا می کند
- در این حالت  $\text{queue\_full}$  باید تمام صف را به چپ شیفت دهد به گونه ای که عنصر اول صف در مکان صفر ارایه قرار گیرد  $\text{front}$  برابر 1- و  $\text{rear}$  به صورت مناسب تصحیح شود.
- شیفت زمان گیر پیچیدگی زمانی  $O(\text{MAX\_QUEUE\_SIZE})$  برابر  $\text{queue\_full}$  است.

## صف ها به عنوان یک نوع داده مجرد

## ▪ پیاده سازی ۲

از یک ارایه 1D استفاده کرده و آن را به صورت حلقوی در نظر بگیریم.

**queue[]**



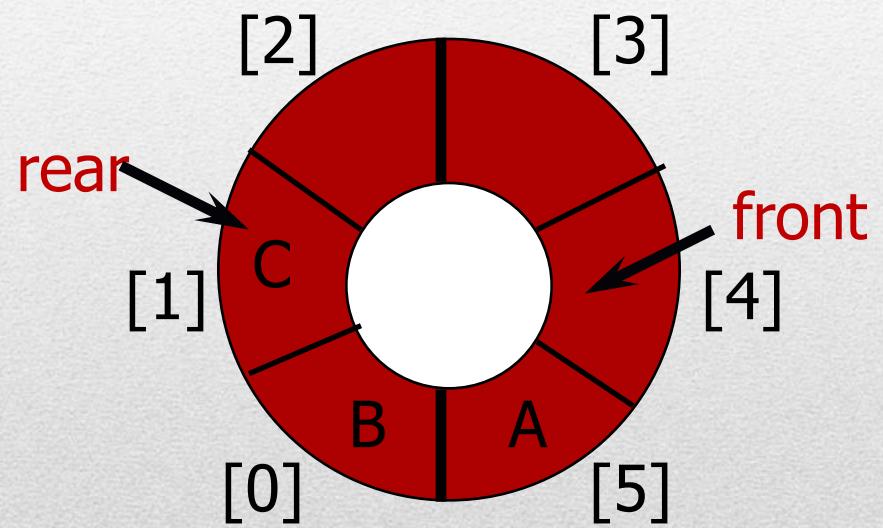
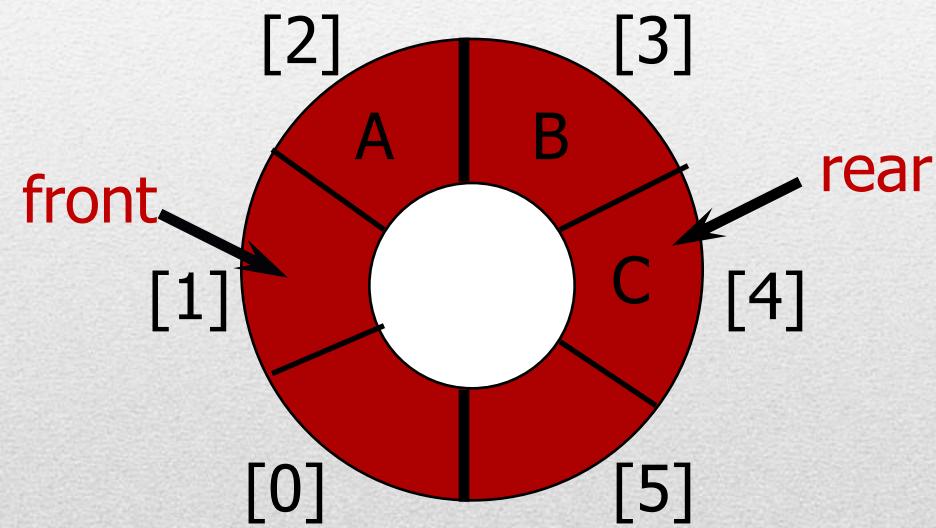
# صف حلقوی

یک محل قبل از اولین عنصر

front

محل آخرین عنصر

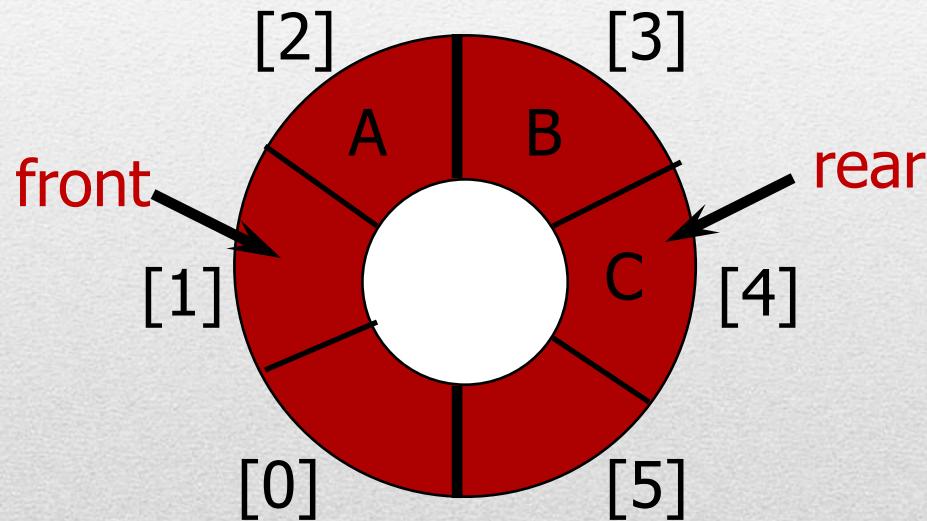
rear



# صف حلقوی

## ▪ اضافه کردن به صف

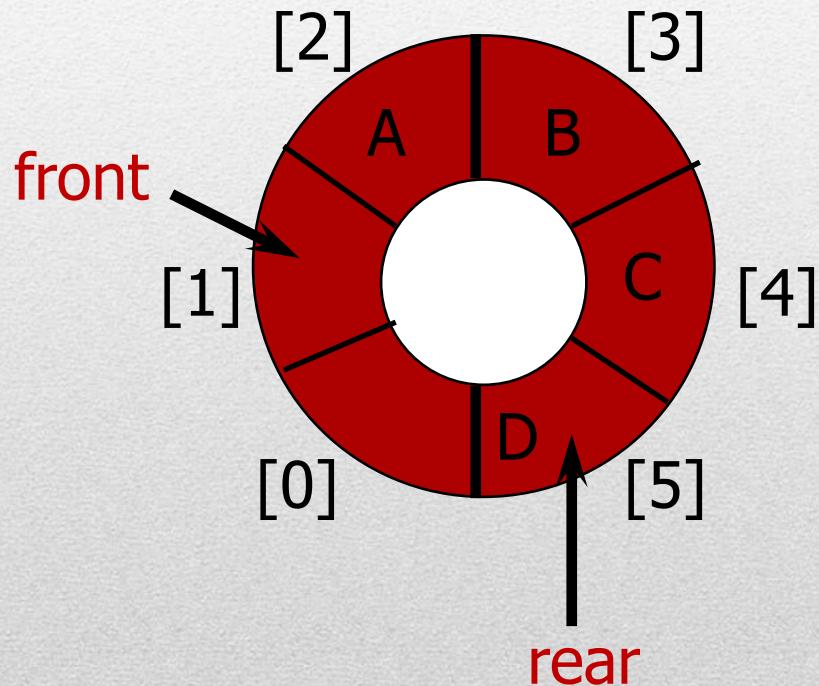
rear را در جهت عقربه های ساعت یک واحد افزایش دهید



# صف حلقوی

## ▪ اضافه کردن به صف

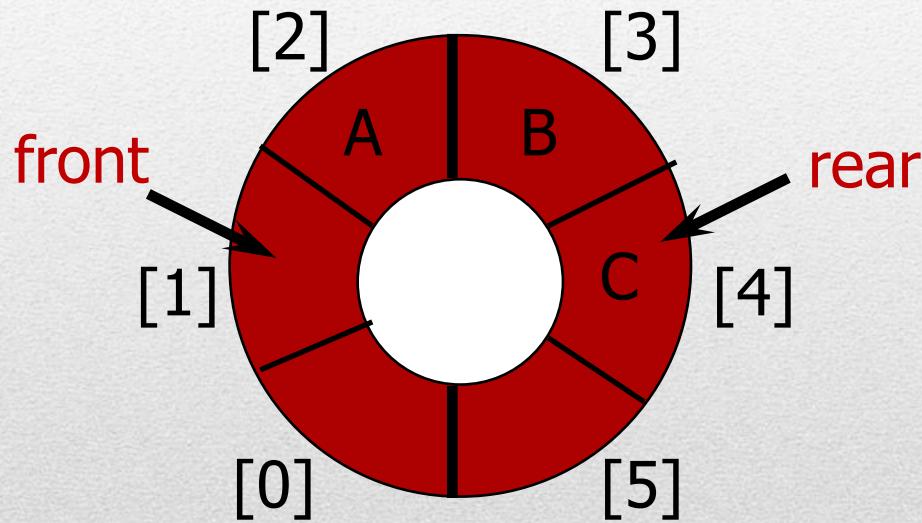
rear را در جهت عقربه های ساعت یک واحد افزایش دهید  
عنصر مورد نظر را در محل queue[rear] قرار دهید



## صف حلقوی

## ▪ حذف کردن یک عنصر

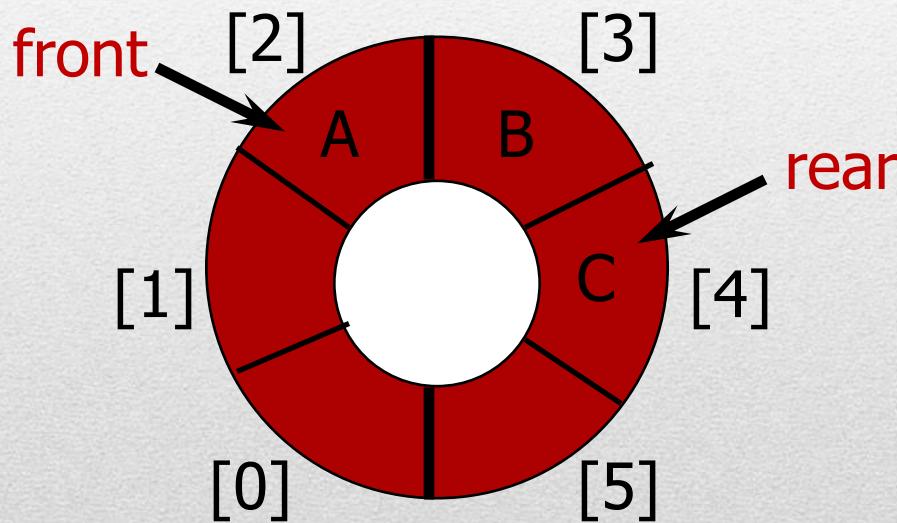
front را در جهت عقربه های ساعت یک واحد افزایش دهید



# صف حلقوی

## ▪ حذف کردن از صف

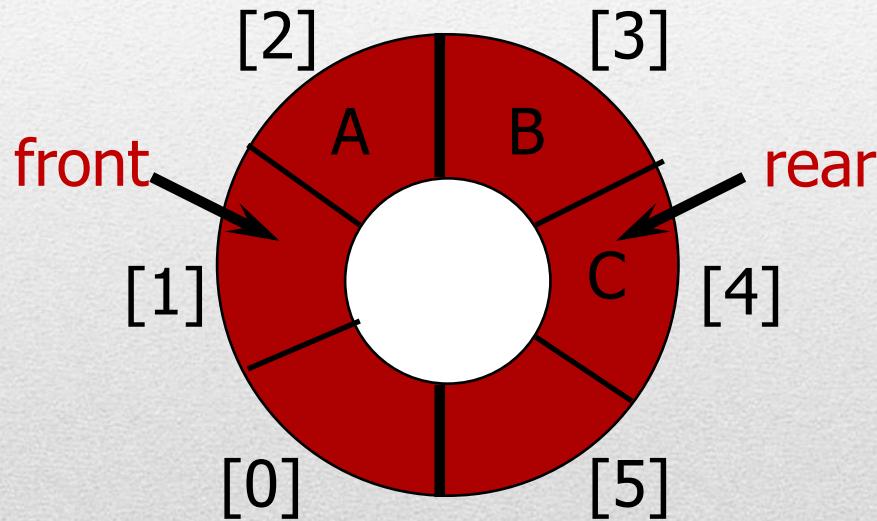
front را در جهت عقربه های ساعت یک واحد افزایش دهید  
عنصر مورد نظر را از محل queue[front] استخراج کنید



# صف حلقوی

▪ اضافه کردن به rear در جهت عقربه های ساعت

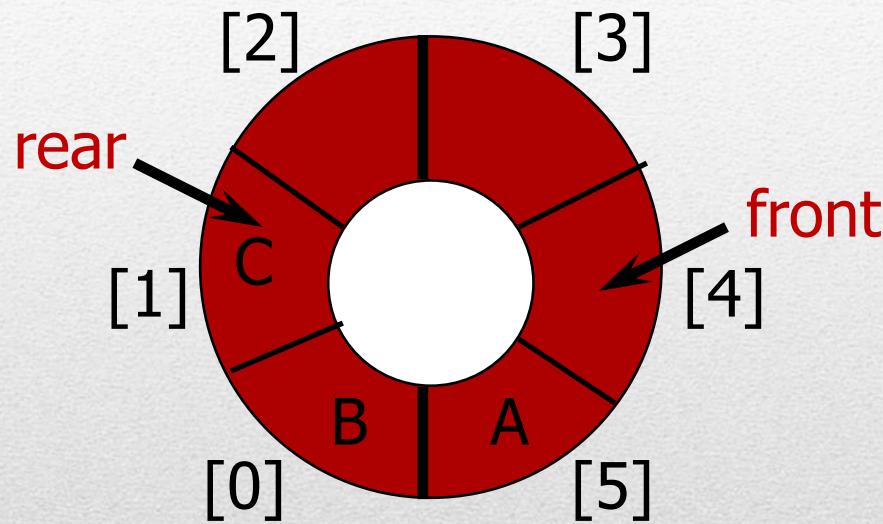
```
rear++;  
if (rear == capacity) rear = 0;
```



```
rear = (rear + 1) % capacity;
```

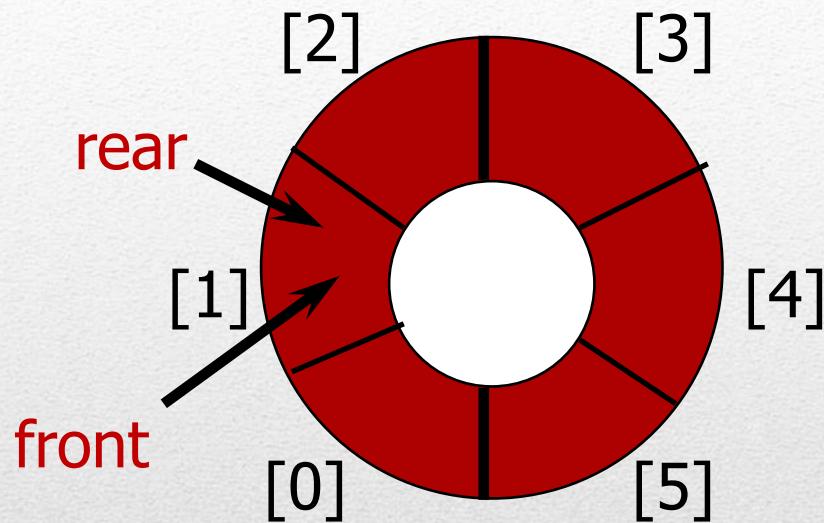
## صف حلقوی

▪ صف خالی



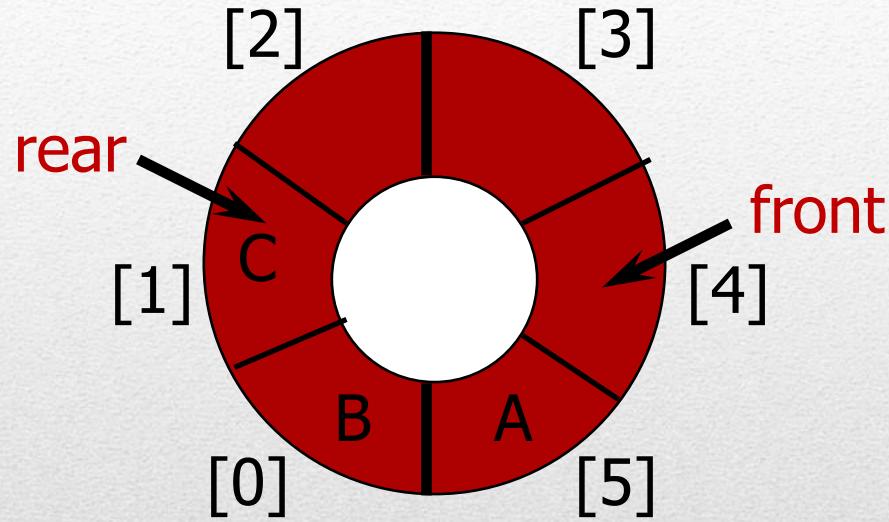
# صف حلقوی

## ▪ صف خالی



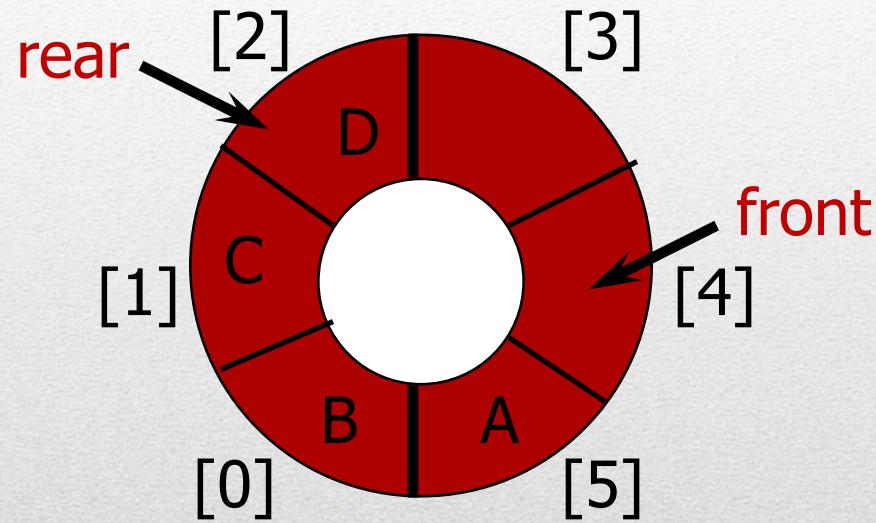
- پس از چند عمل حذف صف خالی می شود، هنگامی که یک صف ایجاد می شود خالی است
- $\text{front} = \text{rear} = 0$

# صف حلقوی



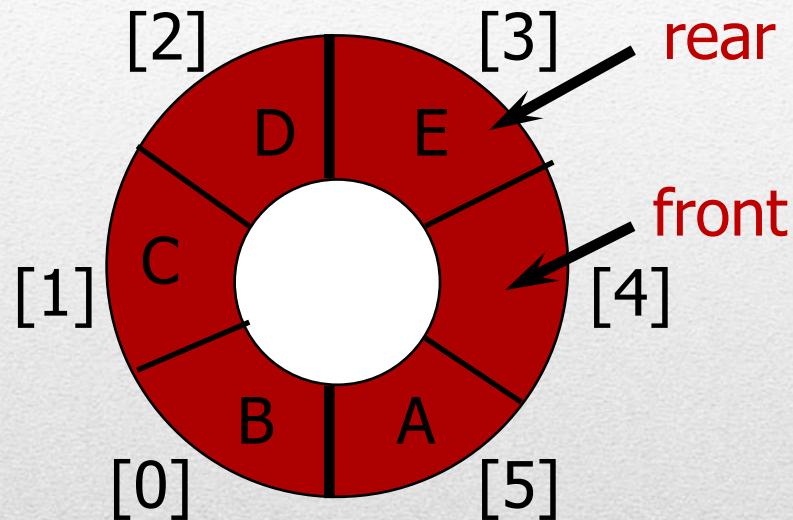
# صف حلقوی

صف پر ■



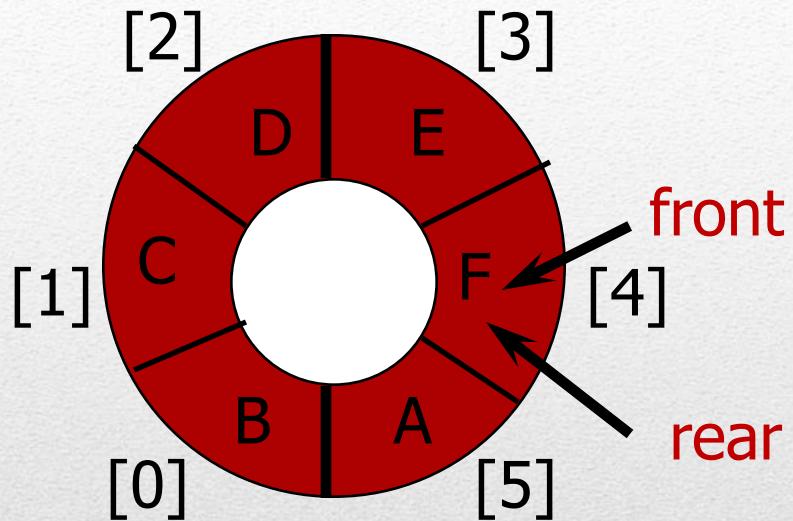
صف حلقوی

▪ صف پر



# صف حلقوی

## ▪ صف پر



- هنگامی که یک سری عنصر به صف اضافه شود تا این صف پر شود این عمل باعث می شود  $\text{front}=\text{rear}$  شود
- بنابراین نمی توان بین صف خالی و پر تمايز قائل شد

# صف حلقوی

- تمايز بين صف پر و خالي
- اجازه ندهيم صف به طور كامل پر شود
- اگر اضافه کردن يك عنصر باعث پر شدن صف مي شود طول ارييه را اضافه کنيم يا اين عنصر را اضافه نکنيم.

- يك متغير بولی lastOperationIsPush تعريف کنيم
- پس از هر اضافه کردن اين متغير true مي شود.
- پس از هر حذف کردن اين متغير false مي شود.

Queue is empty if ( $front == rear$ ) && !lastOperationIsPush  
Queue is full if ( $front == rear$ ) && lastOperationIsPush

# صف حلقوی

## ▪ تمایز بین صف پر و خالی

یک متغیر صحیح `size` تعریف کنیم

▪ بعد از هر عمل اضافه کردن `size++`

▪ بعد از هر عمل حذف کردن `size--`

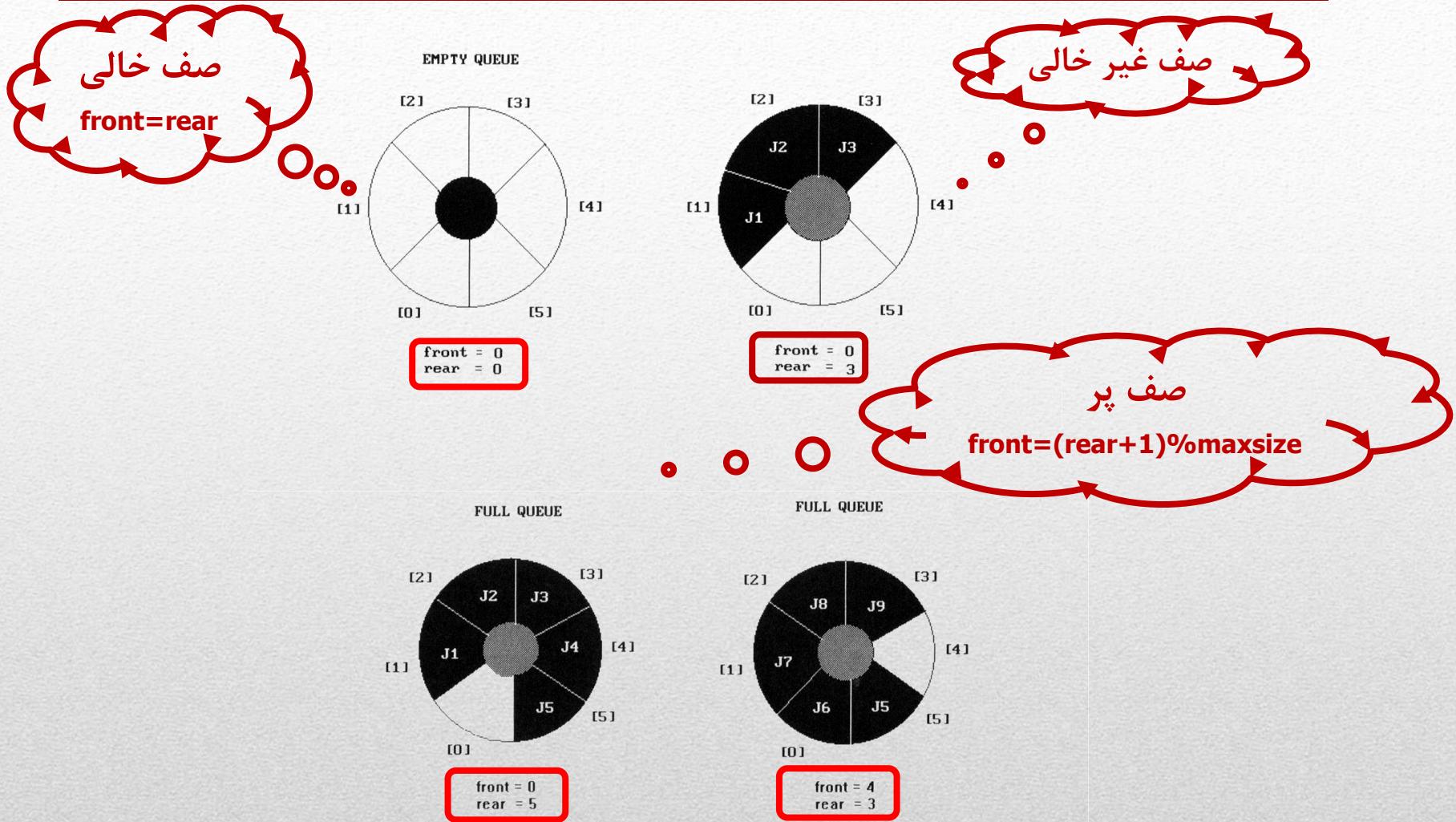
▪ Queue is empty if (`size == 0`)

▪ Queue is full if (`size == arrayLength`)

کارایی هنگامی که از راهکار اول استفاده شود بیشتر است

# صف حلقوی

# صف حلقوی



مشکل: یک فضای خالی باقی می ماند

```
void addq(int front, int *rear, element item)
{
    /* add an item to the queue */
    *rear = (*rear+1) % MAX_QUEUE_SIZE;
    if (front == *rear) {
        queue_full(rear); /* reset rear and print error*/
        return;
    }
    queue[*rear] = item;
}
```

---

```
element deleteq(int *front, int rear)
{
    element item;
    /* remove front element from the queue and put it in
item */
    if (*front == rear)
        return queue_empty(); /* queue_empty returns an
error key */
    *front = (*front+1) % MAX_QUEUE_SIZE;
    return queue[*front];
}
```

---

# صف حلقوی

- به گونه ای زوجهای  $(u,v)$  را بباید که پرانتز چپ موجود در محل  $u$  با پرانتز راست محل  $v$  متناظر باشد

$$(((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)$$
$$(2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)$$

$$(a+b)) * ((c+d)0,4$$

پرانتز راست موقعیت ۵ پرانتز چپ متناظر ندارد

پرانتز چپ موقعیت ۷ پرانتز راست متناظر ندارد

$$(8,12)$$

## تطبیق پرانتزهای یک عبارت

## ▪ تمرين

رشته را از چپ به راست پیمایش کرده و هنگامی که یک پرانتز چپ مشاهده شد محل ان را در پشتہ اضافه کنید. هنگامی که یک پرانتز راست مشاهده شد موقعیت پرانتز چپ متناظر ان را از پشتہ خارج کنید

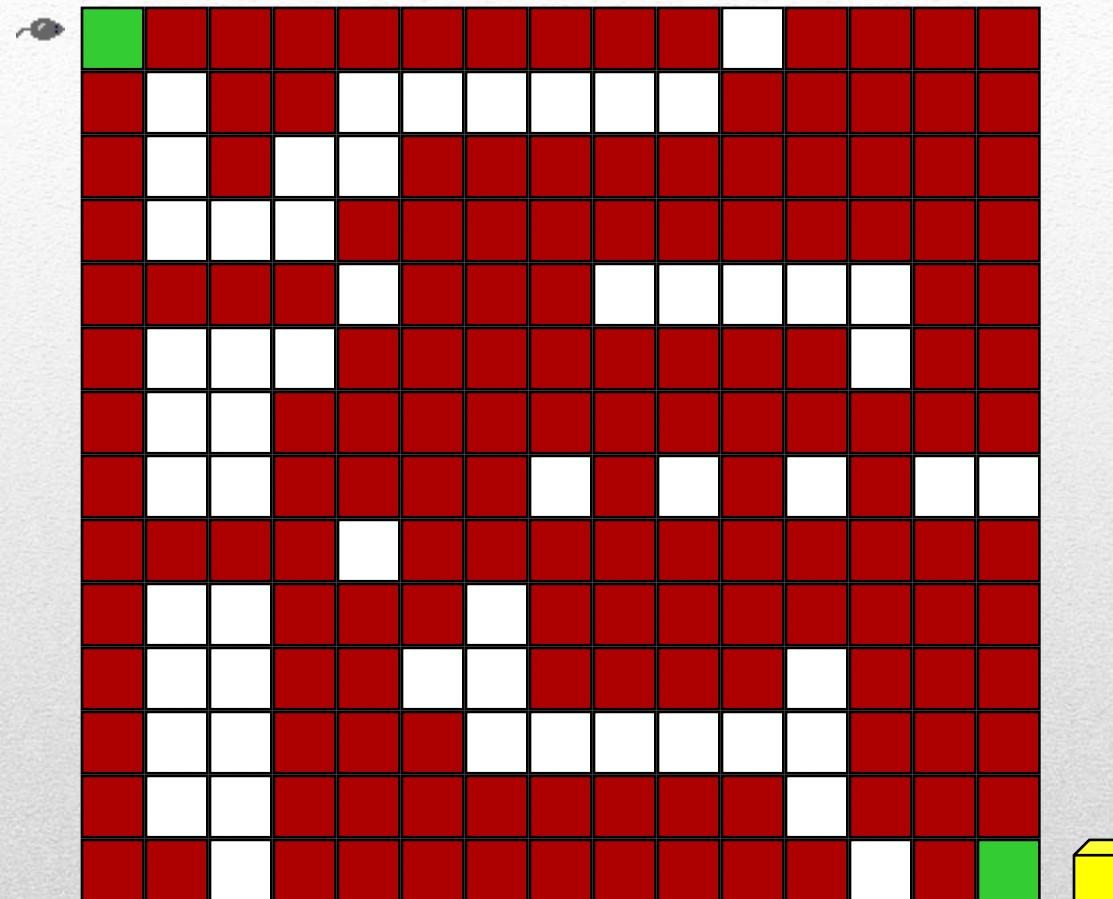
# تطبیق پرانتزهای یک عبارت

▪ (((a+b)\*c+d-e)/(f+g)-(h+j)\*(k-l))/(m-n)

2  
1  
0

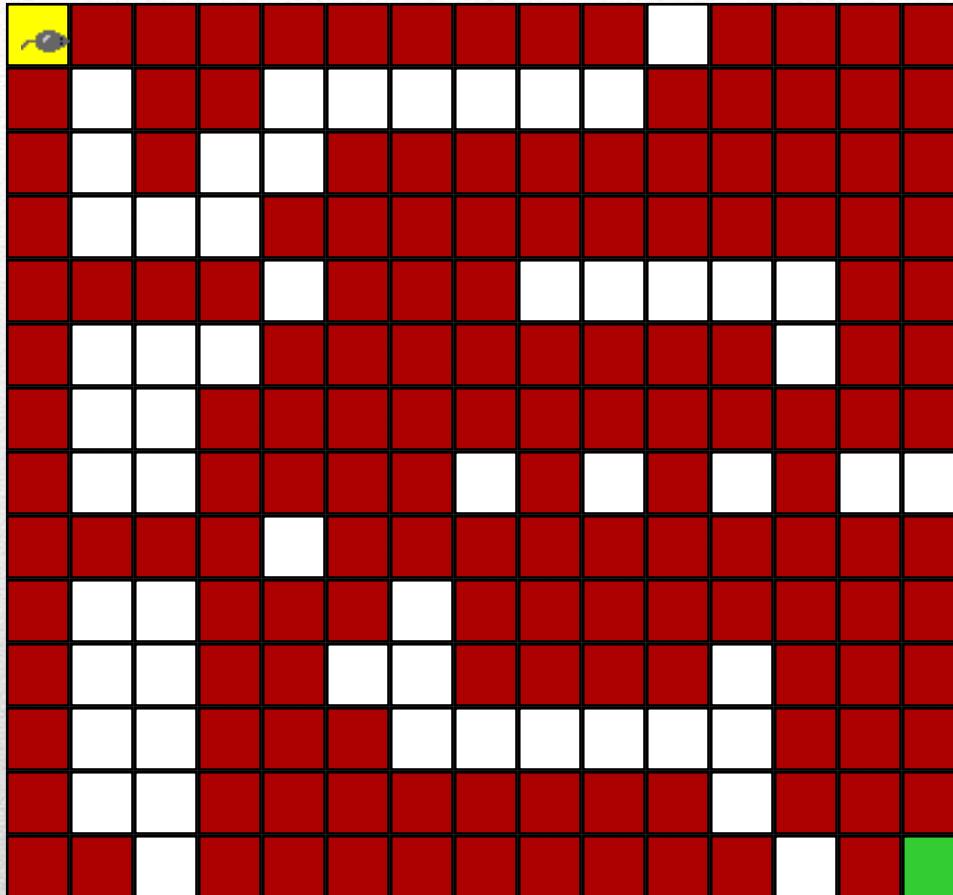
(2,6) (1,13)(15,19) (21,25) (27,31) (0,32)

مثال



# Maze بازی

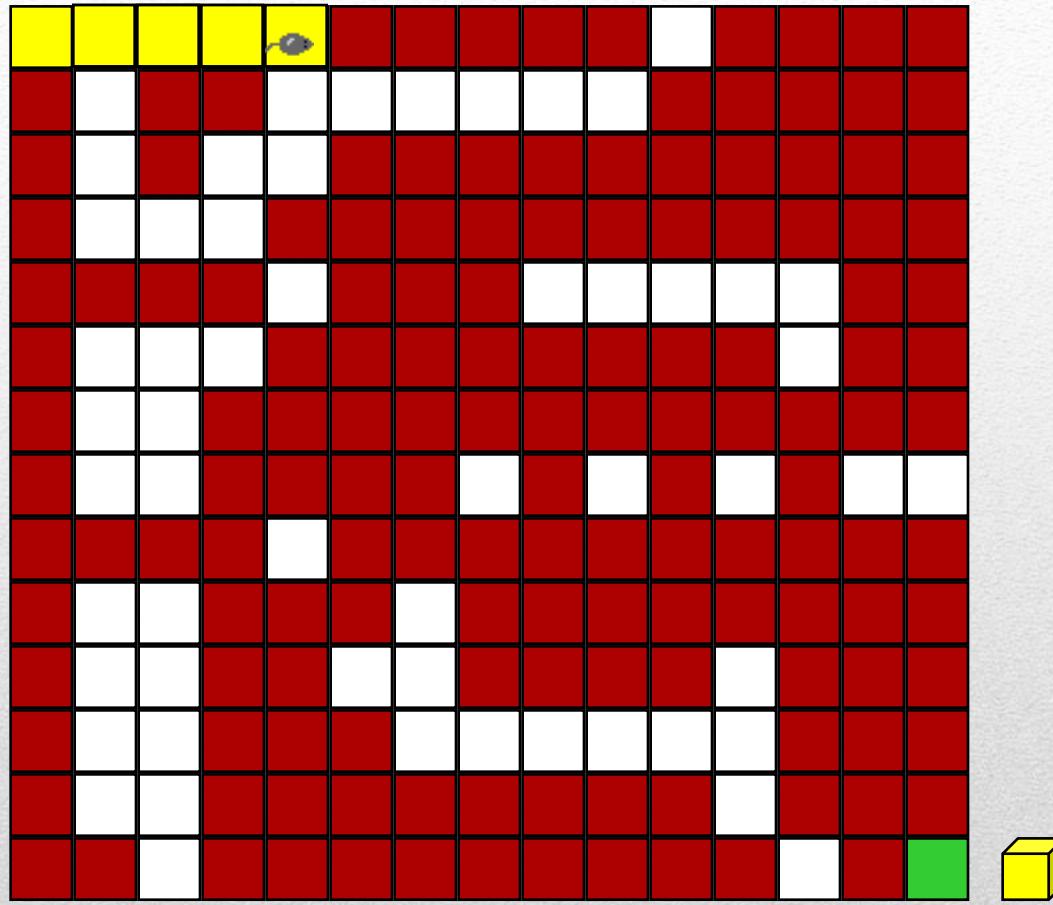
E-mail: Hadi.khademi@gmail.com



▪ حرکات مجاز: راست، پایین، چپ، بالا

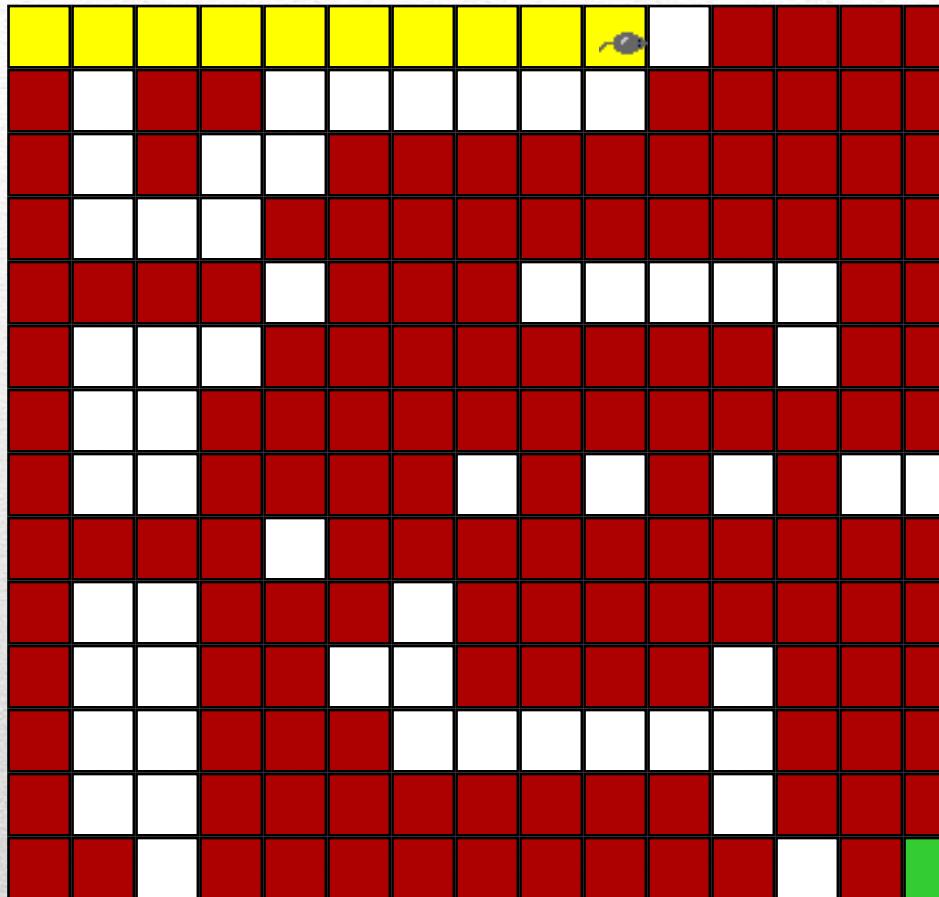
▪ خانه ها را علامت گذاری کنید تا از بازدید مجدد جلوگیری شود.

# Maze بازی



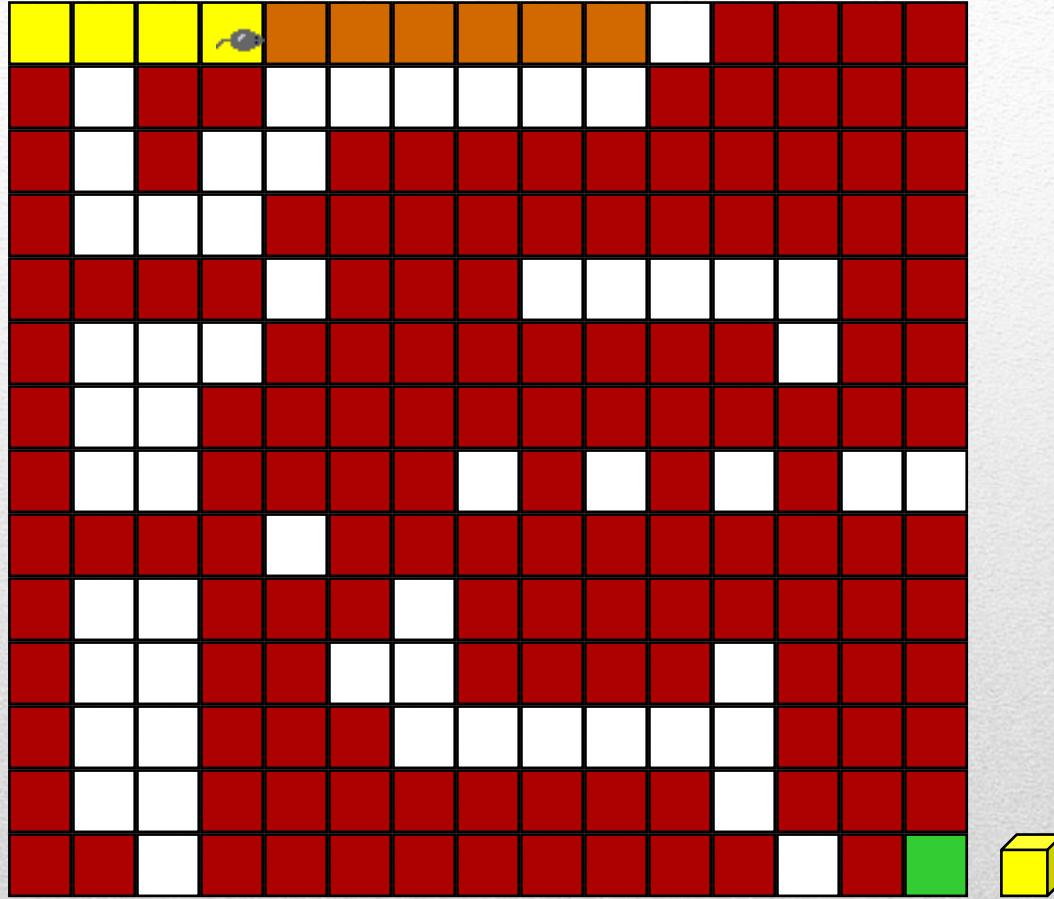
# Maze بازی

E-mail: Hadi.khademi@gmail.com



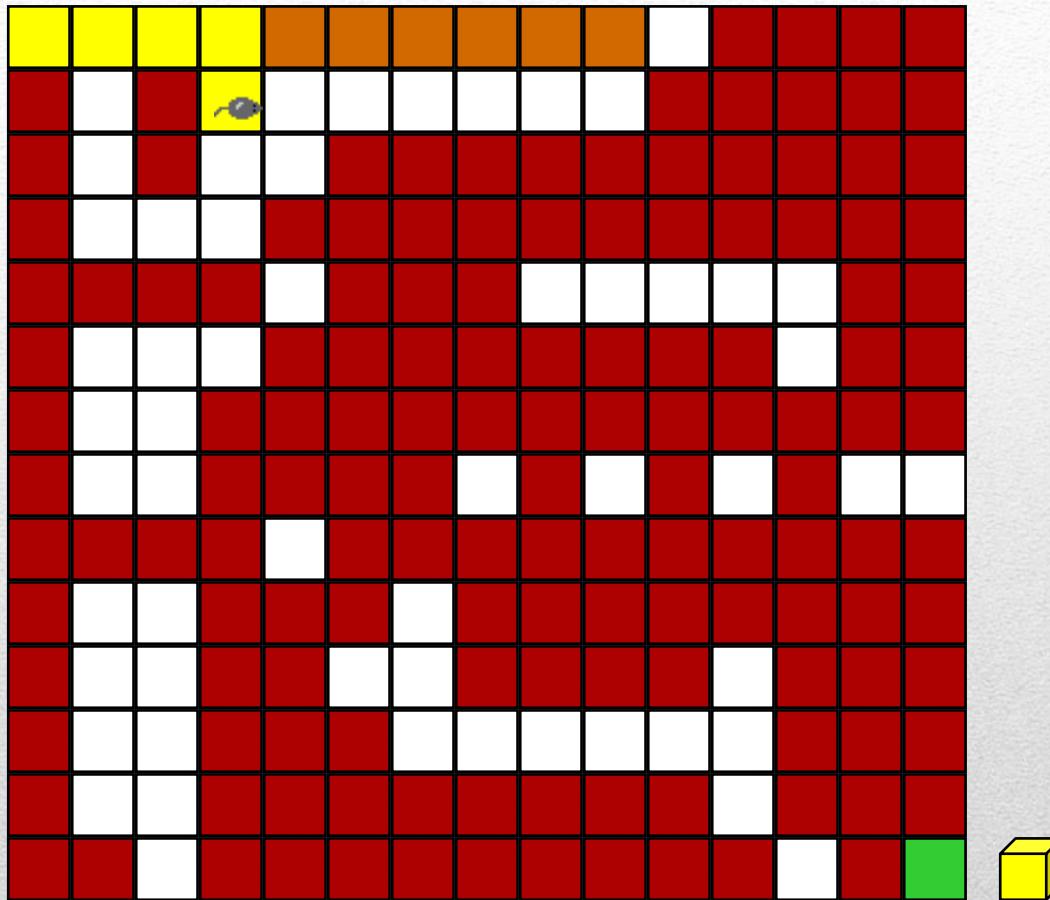
# Maze بازی

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



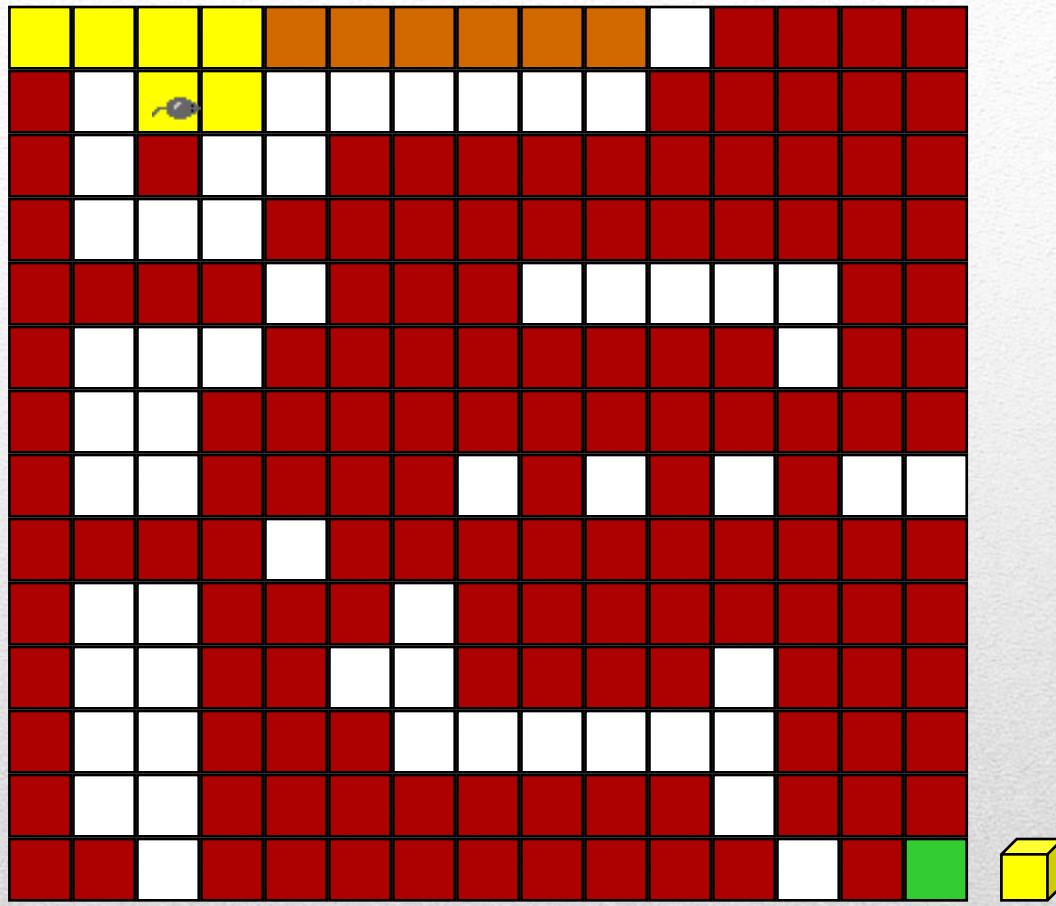
# Maze بازی

▪ به سمت پایین حرکت کنید.



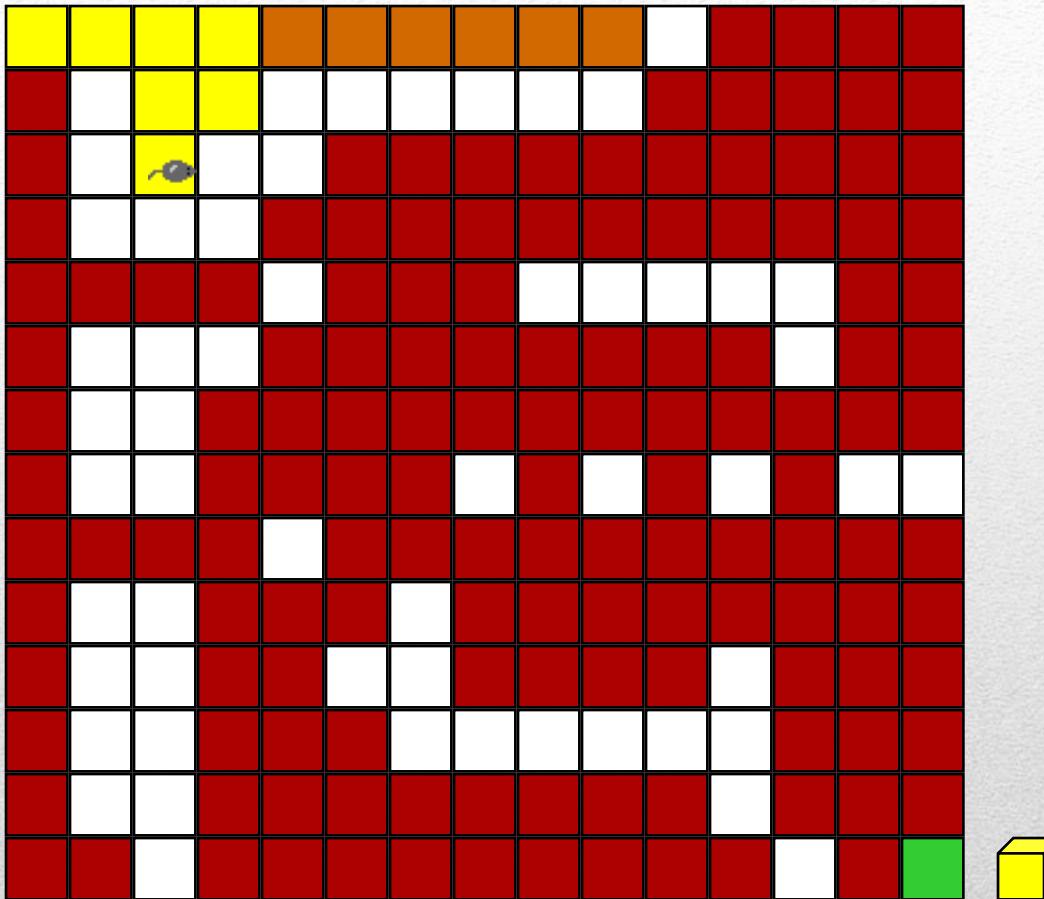
# Maze بازی

▪ به سمت چپ حرکت کنید.



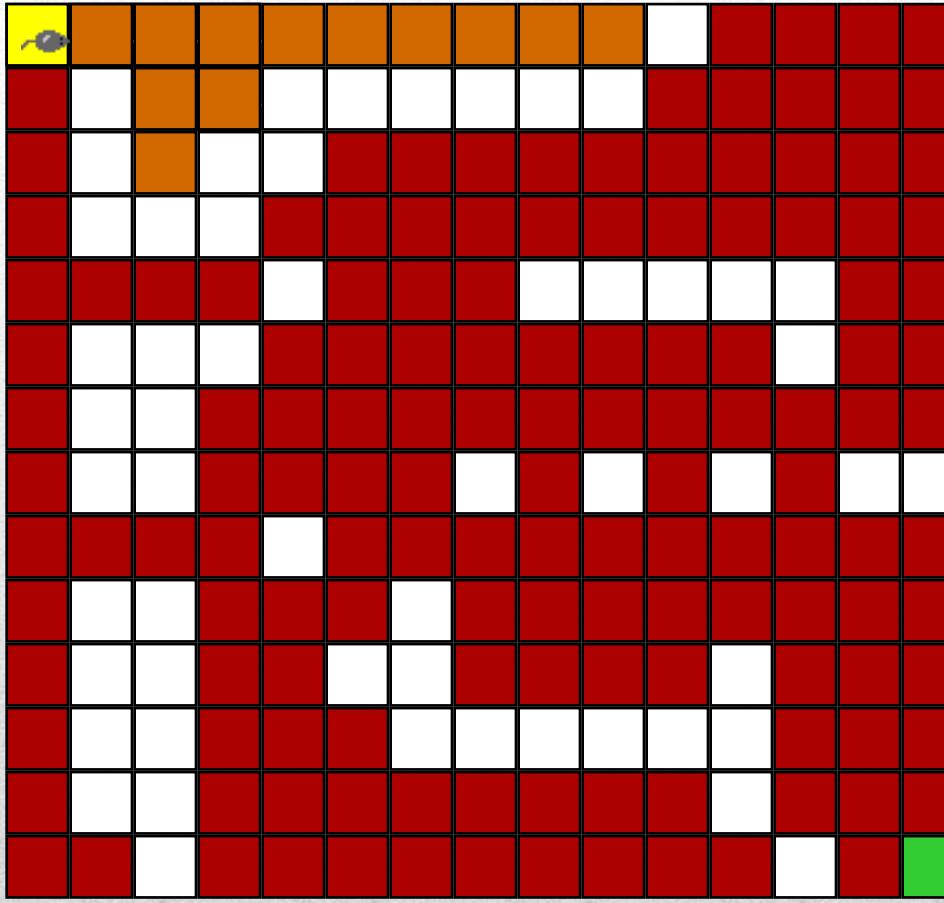
# Maze بازی

▪ به سمت پایین حرکت کنید.



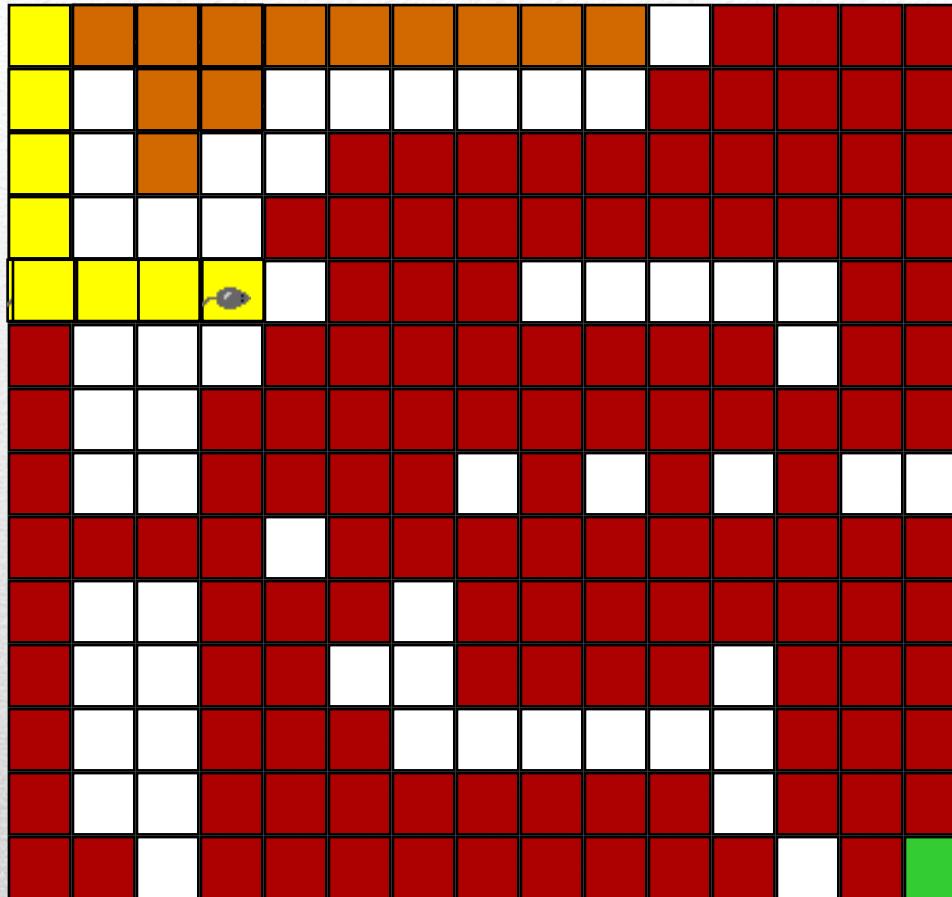
# Maze بازی

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



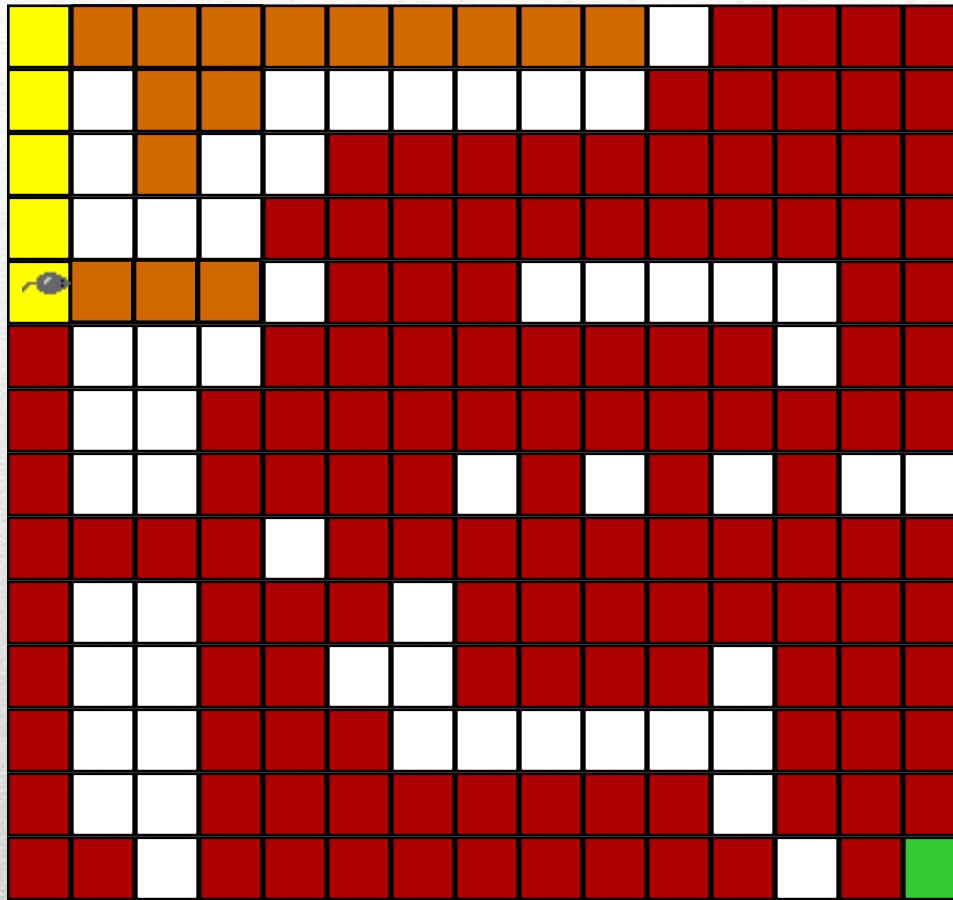
- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد.
- به سمت پایین حرکت کنید.

# Maze بازی



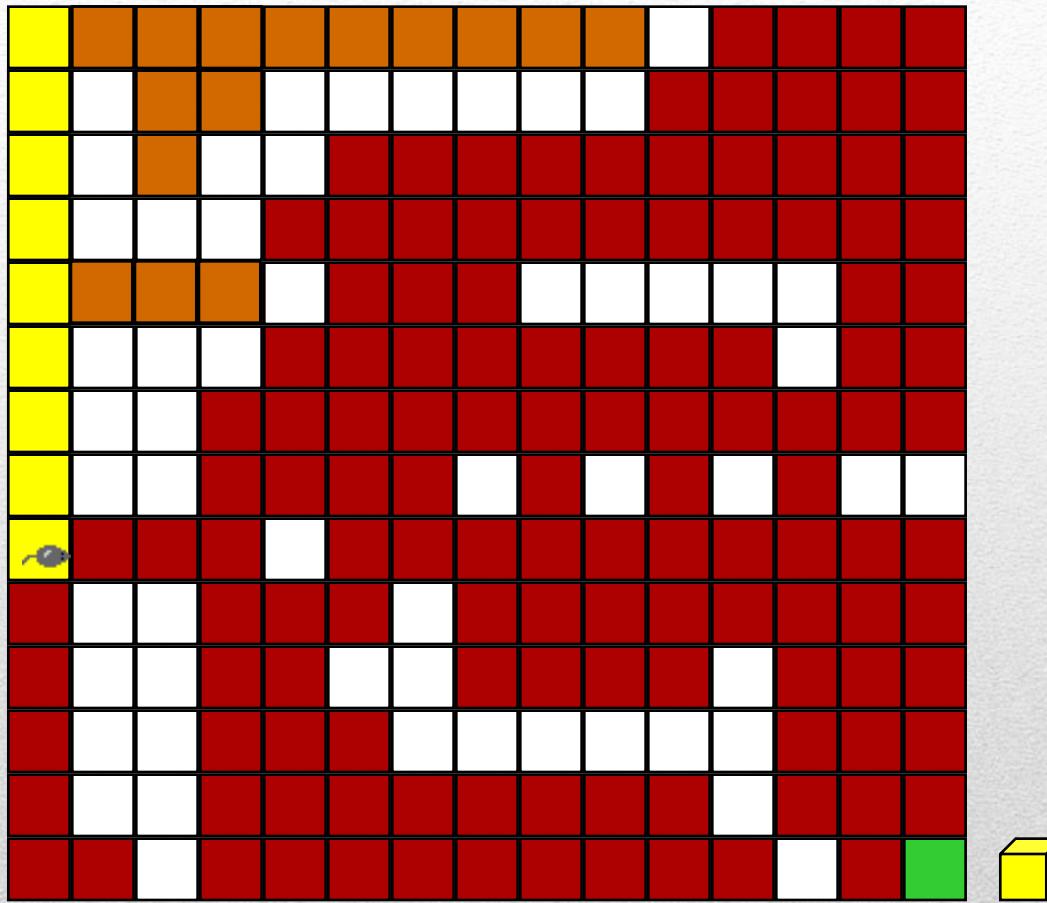
# Maze بازی

- به سمت راست حرکت کنید.
- عقبگرد.



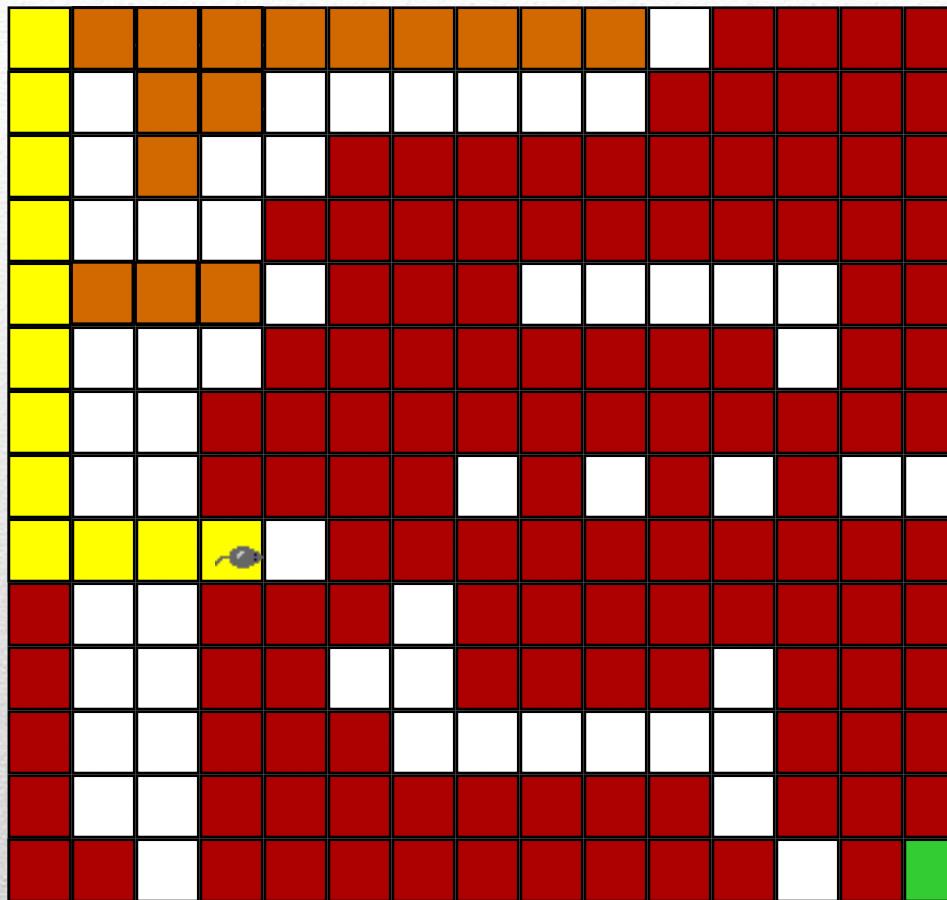
# Maze بازی

▪ به سمت پایین حرکت کنید.



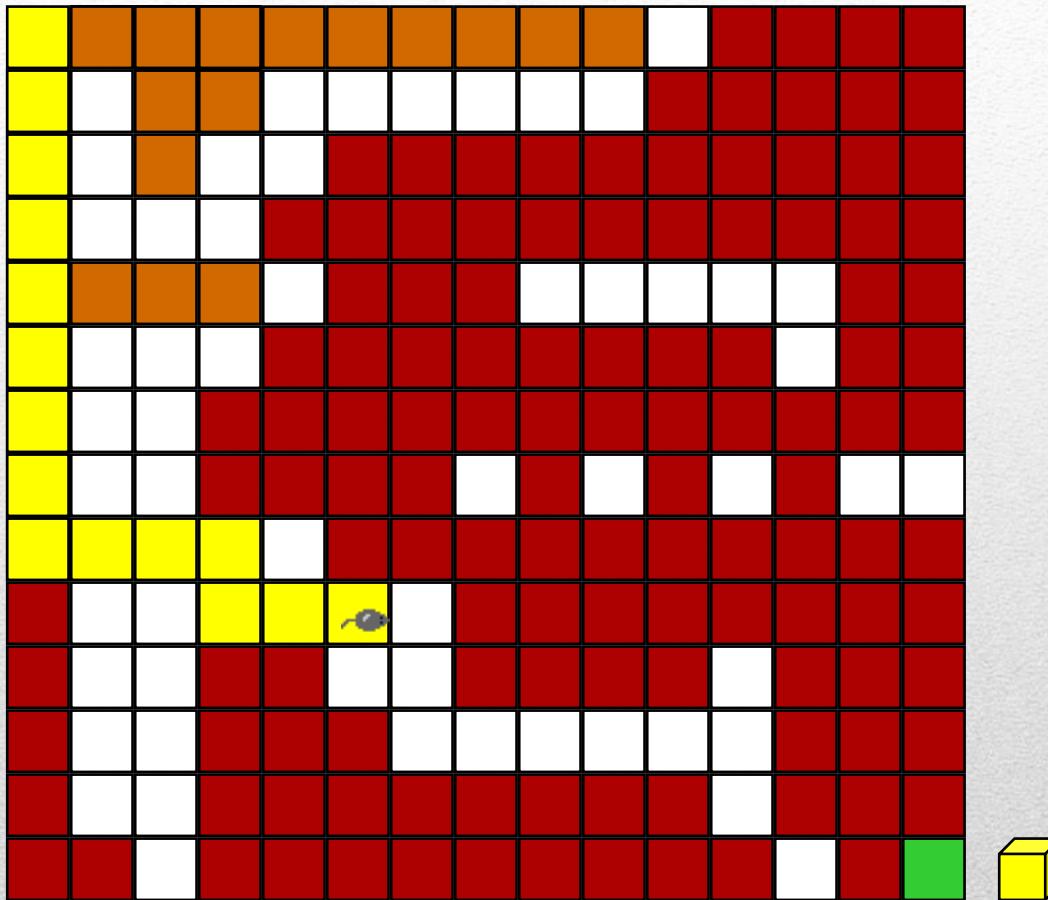
# Maze بازی

▪ به سمت راست حرکت کنید.

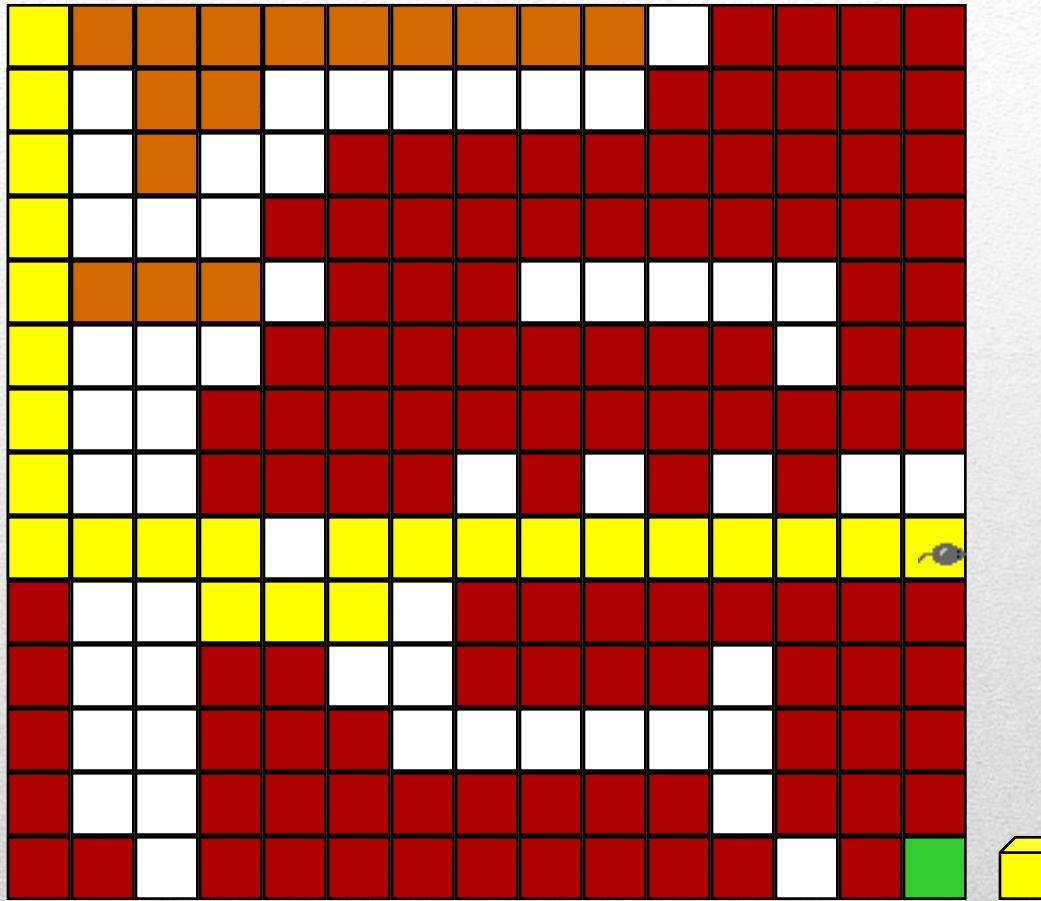


▪ یکی به سمت پایین و سپس به راست حرکت کنید.

# Maze بازی

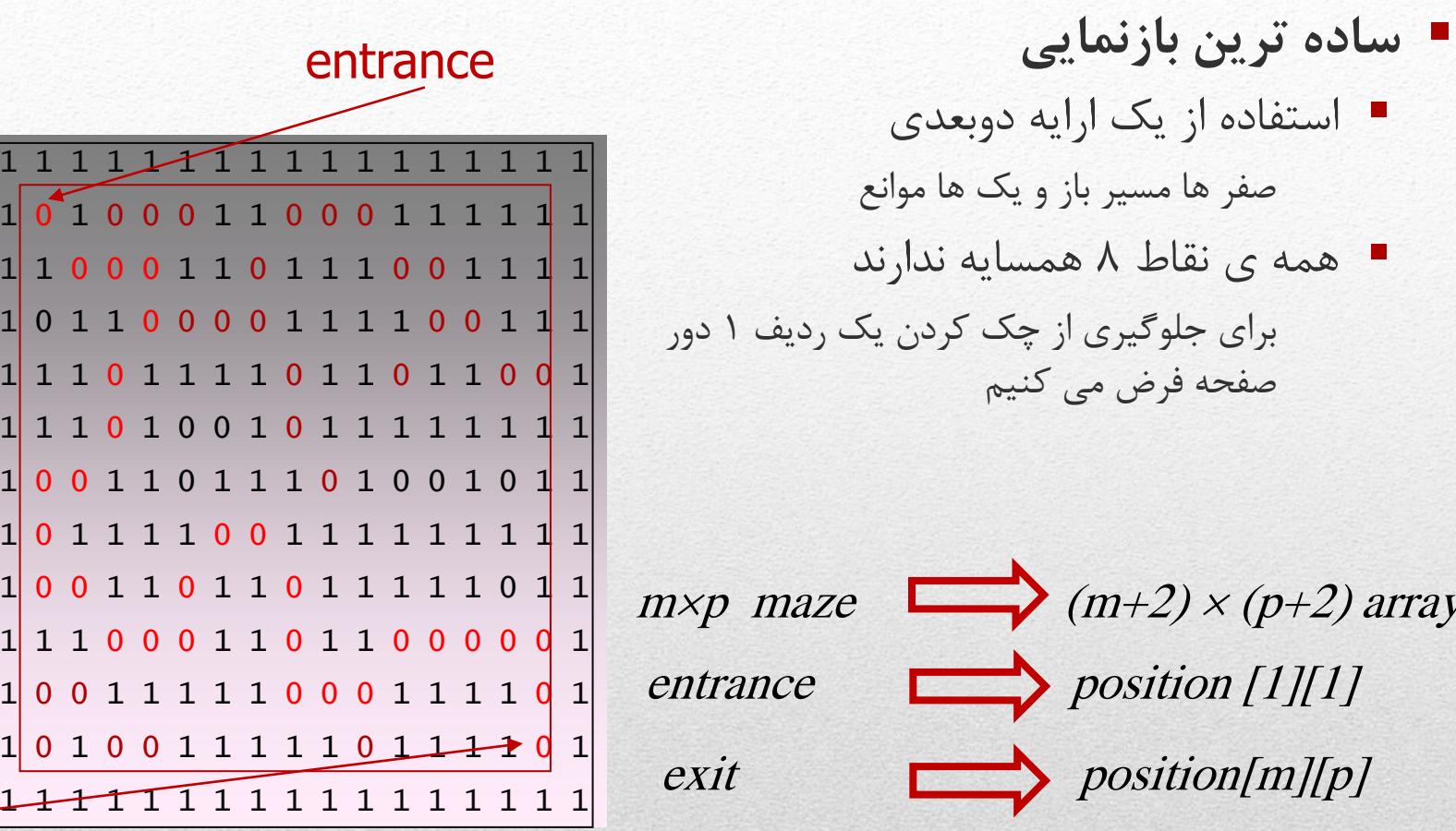


▪ یکی به سمت بالا و سپس به راست حرکت کنید.



# Maze بازی

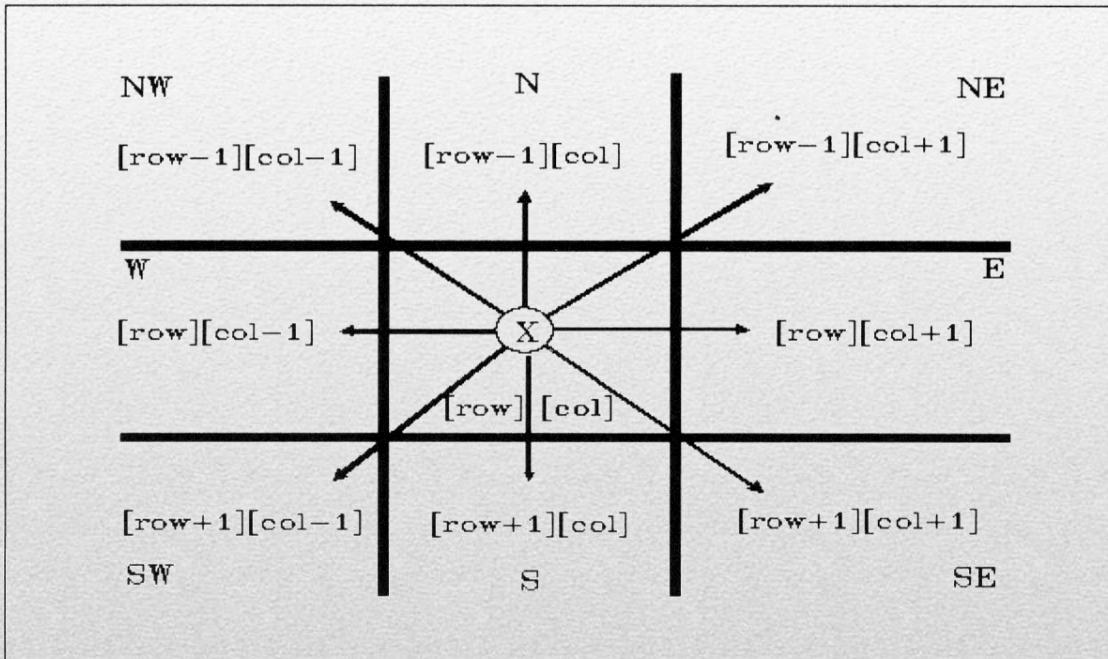
- به سمت پایین حرکت کرده و سپس از Maze خارج شوید.
- مسیر از شروع تا به پایان بر روی پشته قرار دارد.



# بازی Maze

E-mail: Hadi.khademi@gmail.com

▪ اگر [maze[row][col]] محل فعلی ما باشد حرکتهای مجاز می تواند به صورت زیر باشد.



# Maze بازی

## ▪ پیاده سازی حرکتهای مجاز

```
typedef struct {  
    short int vert;  
    short int horiz;  
} offsets;
```

offsets move[8]; /\*array of moves for each direction\*/

اگر [maze[row][col] محل فعلی ما باشد با حرکت در جهت dir محل بعدی ما عبارت است از

next\_row = row + move[dir].vert;

next\_col = col + move[dir].horiz;

# Maze بازی

```

initialize a stack to the maze's entrance coordinates and
direction to north;
while (stack is not empty) {
    /* move to position at top of stack */
    <row,col,dir> = delete from top of stack;
    while (there are more moves from current position) {
        <next-row, next-col> = coordinates of next move;
        dir = direction of move;
        if ((next-row == EXIT-ROW) && (next-col == EXIT-COL))
            success;
        if (maze[next-row] [next-col] == 0 && _____
                mark[next-row] [next-col] == 0) {
            /* legal move and haven't been there */
            mark[next-row] [next-col] = 1;
            /* save current position and direction */
            add <row,col,dir> to the top of the stack;
            row = next-row;
            col = next-col;
            dir = north;
        }
    }
    printf("No path found\n");
}

```

از یک ارایه دو بعدی  
برای نگهداری  
 محلهای مشاهده شده تا  
 کنون استفاده می شود.  
 یک پشته برای نگهداری  
 مسیر به کار می رود

```

#define MAX_STACK_SIZE 100
/*maximum stack size*/
typedef struct {
    short int row;
    short int col;
    short int dir;
} element;
element
stack[MAX_STACK_SIZE];

```

# بازی Maze

```

void path(void)
{
/* output a path through the maze if such a path exists */
    int i, row, col, next_row, next_col, dir, found = FALSE;
    element position;
    mark[1][1] = 1; top = 0;
    stack[0].row = 1; stack[0].col = 1; stack[0].dir = 1;
    while (top > -1 && !found) {
        position = delete(&top);
        row = position.row; col = position.col;
        dir = position.dir;
        while (dir < 8 && !found) {
            /* move in direction dir */
            next_row = row + move[dir].vert;
            next_col = col + move[dir].horiz;
            if (next_row == EXIT_ROW && next_col == EXIT_COL)
                found = TRUE;
            else if ( !maze[next_row][next_col] &&
                      ! mark[next_row][next_col]) {
                mark[next_row][next_col] = 1;
                position.row = row; position.col = col;
                position.dir = ++dir;
                add(&top, position);
                row = next_row; col = next_col; dir = 0;
            }
            else ++dir;
        }
    }
    if (found) {
        printf("The path is:\n");
        printf("row col\n");
        for (i = 0; i <= top; i++)
            printf("%2d%5d",stack[i].row, stack[i].col);
        printf("%2d%5d\n",row,col);
        printf("%2d%5d\n",EXIT_ROW,EXIT_COL);
    }
    else printf("The maze does not have a path\n");
}

```

# Maze بازی

R3 C12 D 5
R3 C13 D 6
R2 C12 D 3
R2 C11 D 2
R1 C10 D 3
R1 C9 D 2
R1 C8 D 2
R2 C7 D 1
R3 C6 D 1
R3 C5 D 2
R2 C4 D 3
R1 C5 D 5
R1 C4 D 2
R1 C3 D 2
R2 C2 D 1
R1 C1 D 3

R 1 C 1 D 1

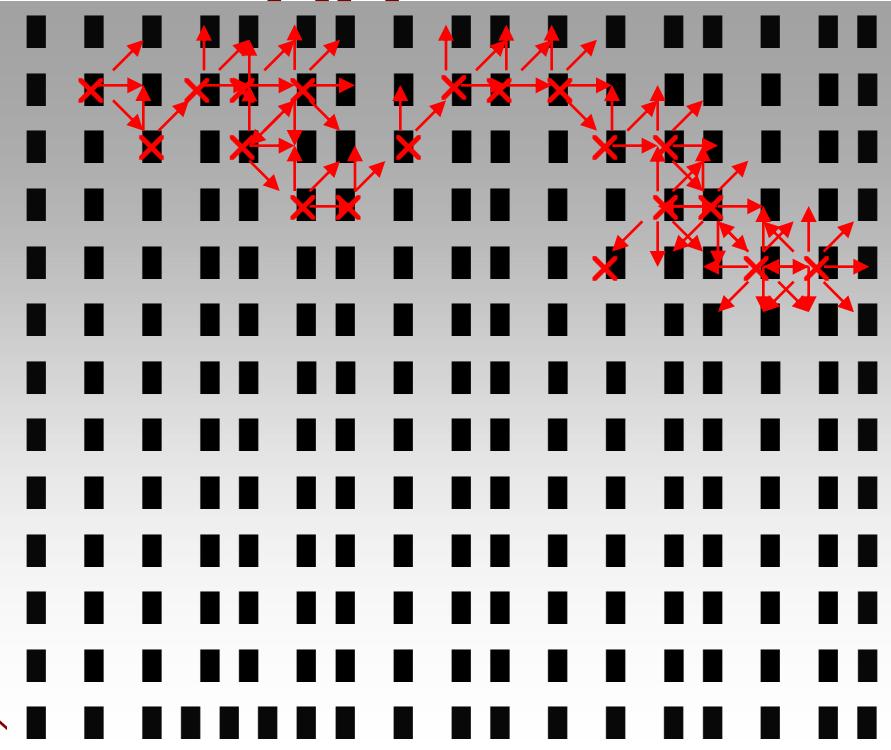
R: row  
C: col  
D: dir

Pop out



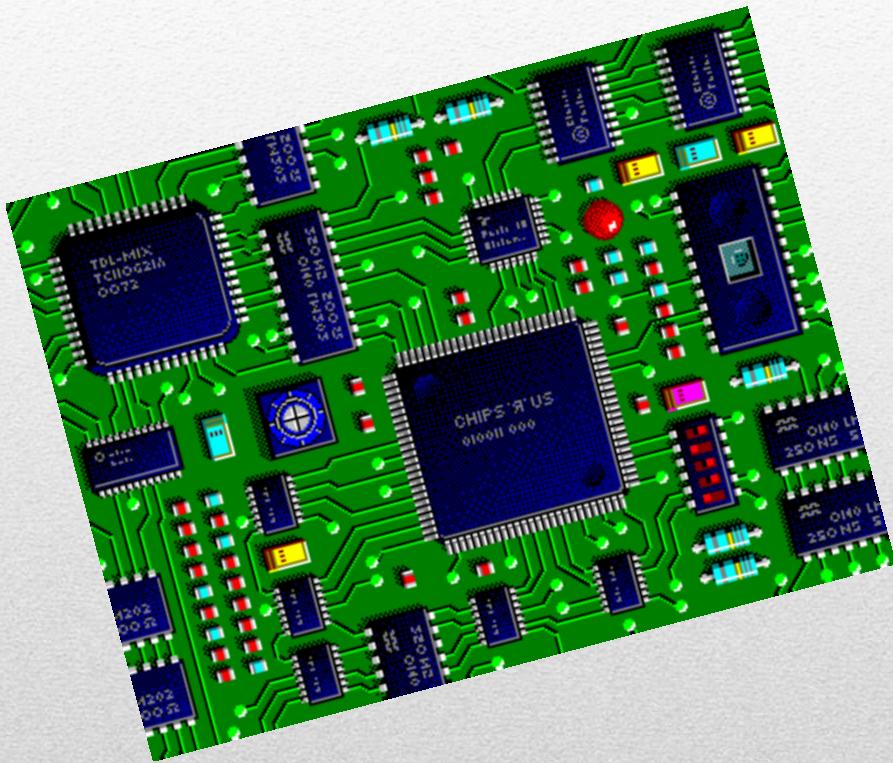
Name	Dir	<i>move[dir].vert</i>	<i>move[dir].horiz</i>
N	0	-1	0
NE	1	-1	1
E	2	0	1
SE	3	1	1
S	4	1	0
SW	5	1	-1
W	6	0	-1
NW	7	-1	-1

Initially set  
mark[1][1]=1



maze[15][11]: exit

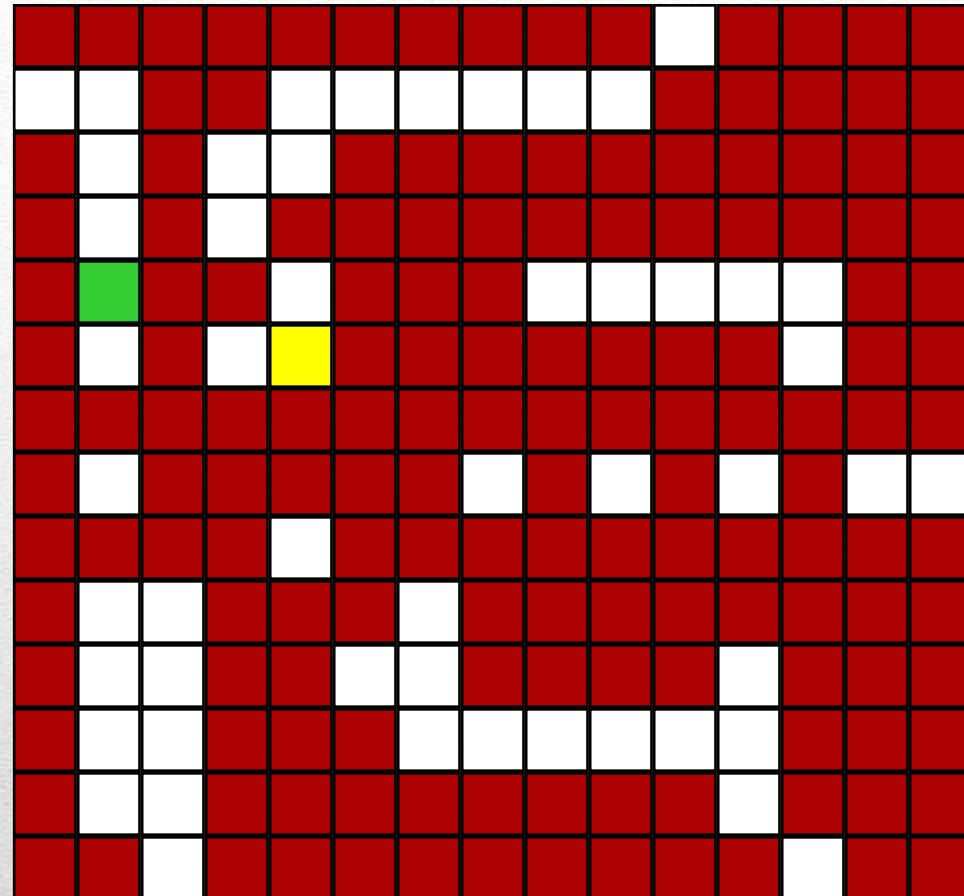
- می توان به جای پشته از صفحه استفاده کرد
- منجر به یافتن کوتاهترین مسیر به خروجی می شود.



# مسیریابی برای سیم

█ start pin

█ end pin



- همه مربع هایی که با فاصله یک واحد از start قابل دسترس هستند را برجسب بزنید. (و این روش را دنبال می کنیم. یعنی دو، سه، ....)

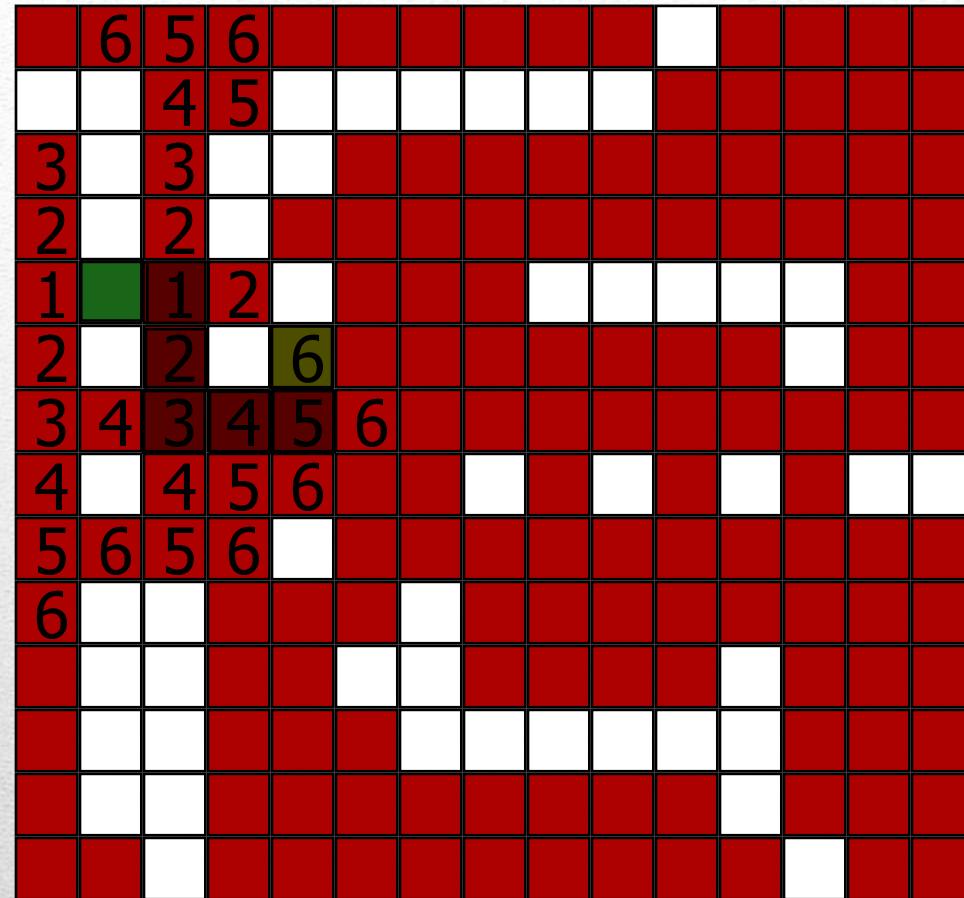
## مسیریابی برای سیم



start pin



end pin



▪ به نقطه مورد نظر رسیده ایم مسیر را باز گردید

# مسیریابی برای سیم

$$a = 4, b = c = 2, d = e = 3$$

$$\text{value of } x = a/b - c + d*e - a*c$$

$$\text{Interpretation 1: } ((4/2)-2)+(3*3)-(4*2) = 0+8+9 = 1$$

$$\text{Interpretation 2: } (4/(2-2+3))*(3-4)*2 = (4/3)*(-1)*2 = -2.66\dots$$

■ 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 <sup>1</sup>	Associativity
()	function call	17	left-to-right
[]	array element		
-> .	struct or union member		
-- ++	increment, decrement <sup>2</sup>	16	left-to-right
-- ++	decrement, increment <sup>3</sup>	15	right-to-left
!	logical not		
-	one's complement		
- +	unary minus or plus		
& *	address or indirection		
sizeof	sizeof (in bytes)		
(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

# اولویت عملگرها در C

- کامپایلرها برای ارزیابی عبارات از بازنمایی میان ترتیب (میانوندی) استفاده نمیکنند و به جای آن بازنمایی پس ترتیب (پسوندی) را به کار می برد

**Postfix:**  
no parentheses,  
no precedence

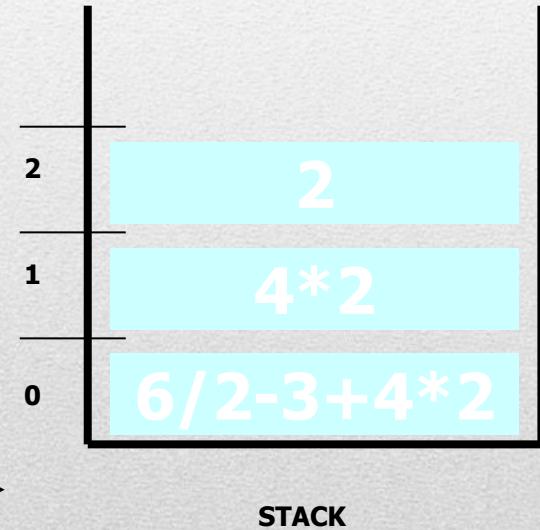
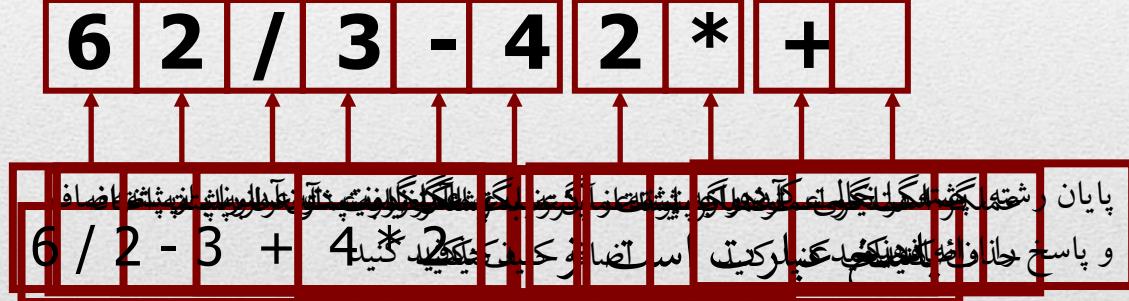
Infix	Postfix
$2+3*4$	$2\ 3\ 4*+$
$a*b +5$	$ab*5+$
$(1+2)*7$	$1\ 2+7*$
$a*b/c$	$ab*c/$
$((a/(b-c+d))* (e-a)*c$	$abc-d+/ea-*c*$
$a/b-c+d*e-a*c$	$ab/c-de*+ac*-$

# ارزیابی عبارات

▪ 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 پشتے قرار دهید



اگر نباشد  $\text{top} = 2$

# ارزیابی عبارات

## ▪ بازنمایی

```
#define MAX_STACK_SIZE 100 /*maximum stack size*/
#define MAX_EXPR_SIZE 100 /*max size of expression*/
typedef enum {lparen ,rparen, plus, minus, times, divide,
              mod, eos, operand} precedence;
int stack[MAX_STACK_SIZE]; /* global stack */
char expr[MAX_EXPR_SIZE]; /* input string */
```

# ارزیابی عبارات

```
precedence get_token(char *symbol, int *n)
{
/* get the next token, symbol is the character
representation, which is returned, the token is
represented by its enumerated value, which
is returned in the function name */
*symbol = expr[(*n)++];
switch (*symbol) {
    case '(' : return lparen;
    case ')' : return rparen;
    case '+' : return plus;
    case '-' : return minus;
    case '/' : return divide;
    case '*' : return times;
    case '%' : return mod;
    case ' ' : return eos;
    default : return operand; /* no error checking,
                                default is operand */
}
}
```

## Get Token

# ارزیابی عبارات

```

int eval(void)
{
/* evaluate a postfix expression, expr, maintained as a
global variable. '\0' is the end of the expression.
The stack and top of the stack are global variables.
get_token is used to return the tokentype and
the character symbol. Operands are assumed to be single
character digits */
precedence token;
char symbol;
int op1, op2;
int n = 0; /* counter for the expression string */
int top = -1;
token = get_token(&symbol, &n);
while (token != eos) {
    if (token == operand)
        add(&top, symbol-'0'); /* stack insert */
    else {
        /* remove two operands, perform operation, and
        return result to the stack */
        op2 = delete(&top); /*stack delete */
        op1 = delete(&top);
        switch(token) {
            case plus: add(&top,op1+op2);
                         break;
            case minus: add(&top, op1-op2);
                         break;
            case times: add(&top, op1*op2);
                         break;
            case divide: add(&top,op1/op2);
                         break;
            case mod: add(&top, op1%op2);
                         break;
        }
    }
    token = get_token(&symbol, &n);
}
return delete(&top); /* return result */
}

```

# ارزیابی عبارات

▪ رشته زیر را به فرم پسوندی تبدیل کنید ■

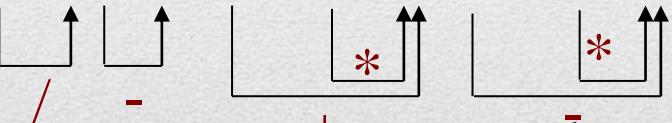
$$a / b - c + d * e - a * c$$

۱) عبارت را به صورت کامل پرانتزگذاری کنید

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

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

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

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


۳) همه ای پرانتزها را حذف کنید.

$$ab / c - de * + ac * -$$

▪ ترتیب عملوندها در فرم میانوندی و پسوندی یکسان است

## تبدیل عبارتهای میانوندی به پسوندی

## ▪ الگوریتم تبدیل رشته میانوندی به پسوندی فرضیات

operators: (, ), +, -, \*, /, %

operands: single digit integer or variable of one character

رشته را از چپ به راست پیمایش کنید.

عملوندها مستقیما در خروجی نوشته می شوند.

عملگرهای داخل پشته مادامیکه اولویت داخل پشته آنها in-stack precedence (isp) بزرگتر و یا مساوی اولویت ورودی عملگر جدید incoming precedence (icp) است از پشته خارج می شوند.

)، دارای isp پایین و icp بالا است.

op	(	)	+	-	*	/	%	eos
Isp	0	19	12	12	13	13	13	0
Icp	20	19	12	12	13	13	13	0

# تبدیل عبارتهای میانوندی به پسوندی

# مثال

$a+^*c$

$abc^*+$

Token	[0]	Stack [1]	[2]	Top	Output
$a$				-1	$a$
$+$	+			0	$a$
$b$	+			0	$ab$
$*$	+	*		1	$ab$
$c$	+	*		1	$abc$
$eos$				-1	$abc^*+$

$a^*(b+c)^*d$

$abc+^*d^*$

Token	[0]	Stack [1]	[2]	Top	Output
$a$				-1	$a$
$*$	*			0	$a$
(	*	(		1	$a$
$b$	*	(		1	$ab$
$+$	*	(	+	2	$ab$
$c$	*	(	+	2	$abc$
)	*			0	$abc +$
$*$	*			0	$abc +^*$
$d$	*			0	$abc +^*d$
$eos$	*			0	$abc +^*d^*$

# تبديل عبارتهای میانوندی به پسوندی

```

void postfix(void)
{
/* output the postfix of the expression. The expression
string, the stack, and top are global */
char symbol;
precedence token;
int n = 0;
int top = 0; /* place eos on stack */
stack[0] = eos;
for (token = get_token(&symbol, &n); token != eos;
     token = get_token(&symbol,&n)) {
    if (token == operand)
        printf("%c",symbol);
    else if (token == rparen) {
        /* unstack tokens until left parenthesis */
        while (stack[top] != lparen)
            print_token(delete(&top));
        delete(&top); /* discard the left parenthesis */
    }
    else {
        /* remove and print symbols whose isp is greater
        than or equal to the current token's icp */
        while(isp[stack[top]] >= icp[token])
            print_token(delete(&top));
        add(&top, token);
    }
}
while ( (token=delete(&top)) != eos)
    print_token(token);
printf("\n");
}

```

**Complexity:  $\Theta(n)$**

*n is the number of tokens in  
the expression*

# تبديل عبارتهای میانوندی به پسوندی