پشته و صف

سید مهدی وحیدی پور

با تشکر از دکتر جواد سلیمی

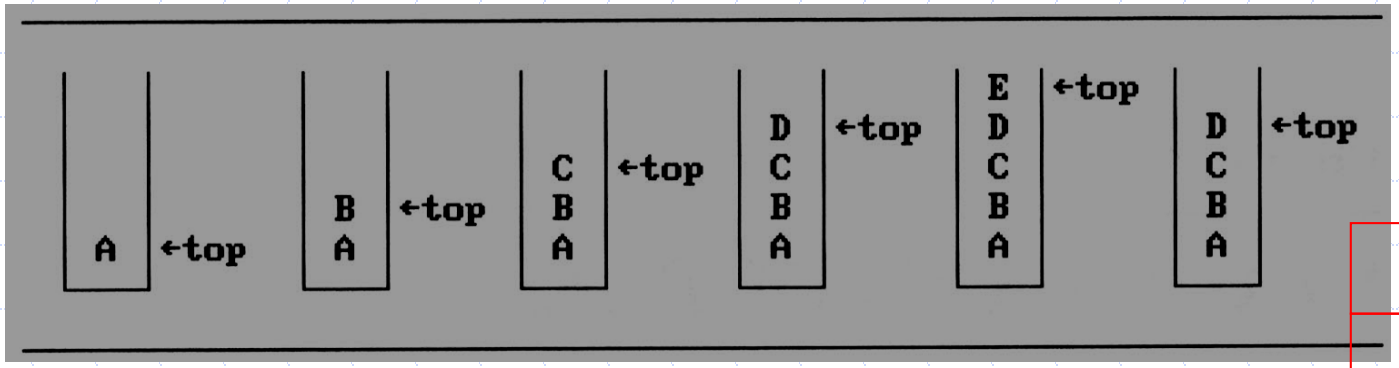
دانشگاه کاشان- دانشکده مهندسی برق و کامپیوتر

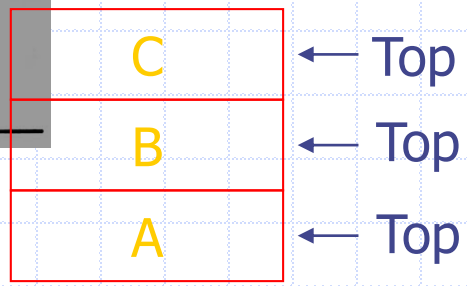
پشته ها و صف ها

- پشته ها به عنوان یک نوع داده مجرد
- صف 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

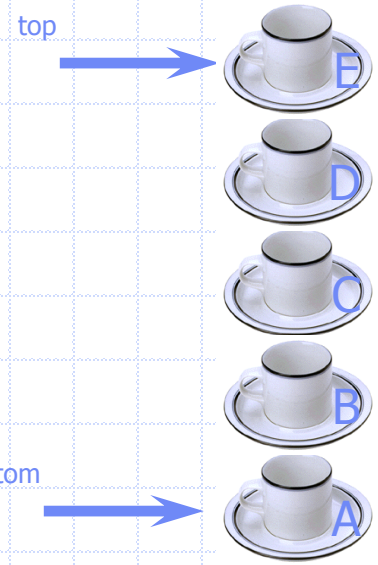
- پشته ها حالت خاصی از لیست های مرتب شده می باشند که جایگذاری و حذف از یک سمت آن که top (بالا) نامیده می شود ، صورت می گیرد.
- پشته یک لیست LIFO (Last-In-First-Out) نیز نامیده می شود.

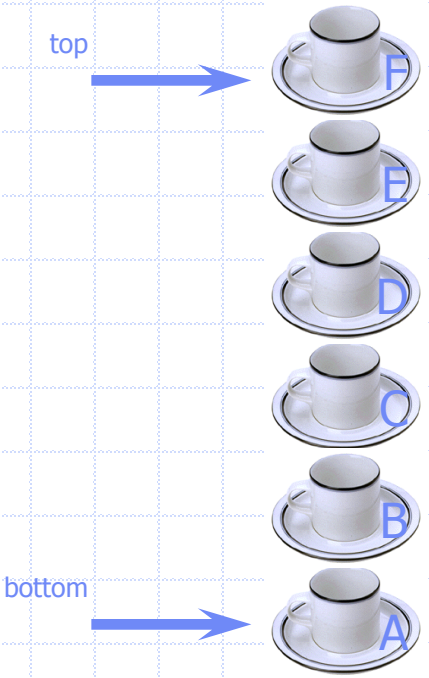




پشته

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





پشته ها به عنوان یک نوع داده مجرد

structure *Stack* is
  objects: a finite ordered list with zero or more elements.
  functions:
    for all *stack* ∈ *Stack*, *item* ∈ *element*, *max_stack_size* ∈ 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.

پشته ها به عنوان یک نوع داده مجرد

- **پیاده سازی:** با کمک ارایه

Stack CreateS(*max_stack_size*) ::=

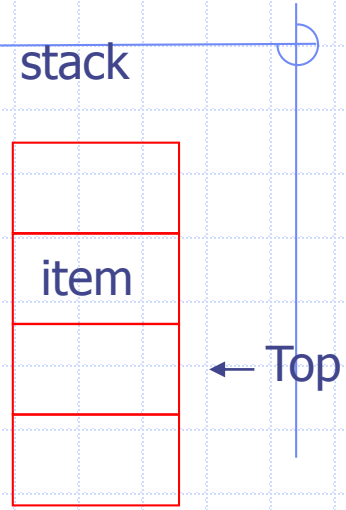
```
#define MAX_STACK_SIZE 100 /*maximum stack size*/
typedef struct {
        int key;
        /* other fields */
        } element;
element stack[MAX_STACK_SIZE];
int top = -1;
```

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

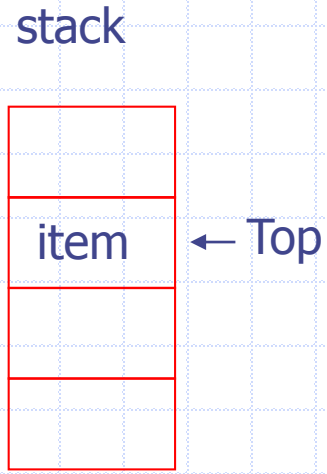
Boolean IsFull(Stack) ::= `top >= MAX_STACK_SIZE-1;`

پشته ها به عنوان یک نوع داده مجرد

```
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;
}
```

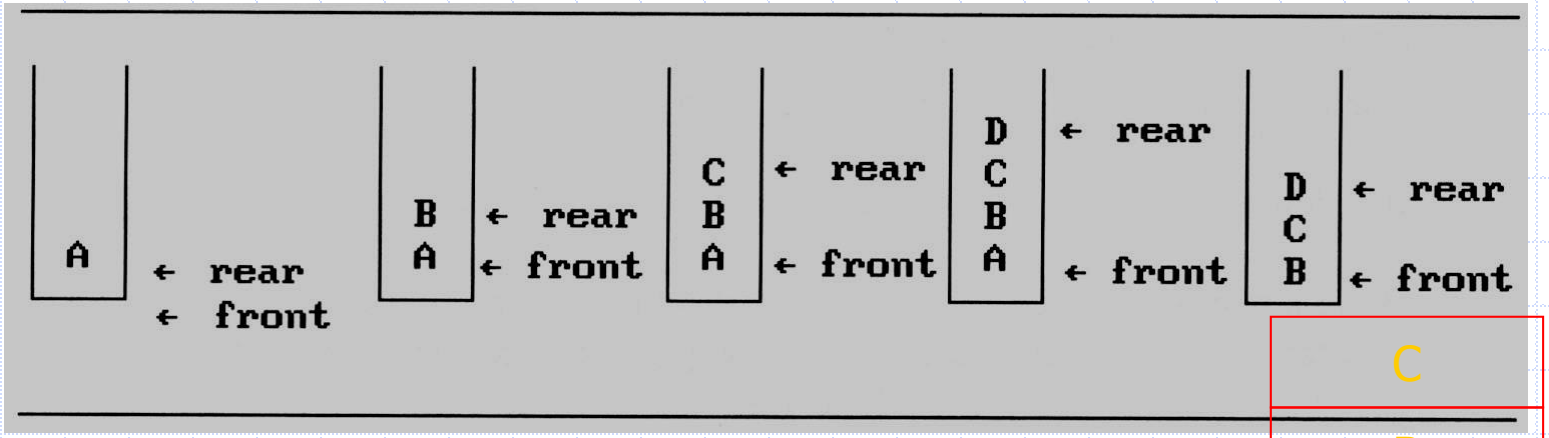


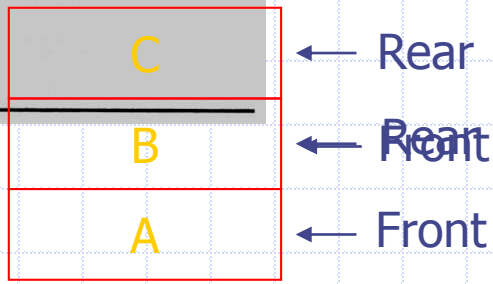

```
element delete(int *top)
{
/* return the top element from the stack */
   if (*top == -1)
      return stack_empty(); /* returns an error key */
   return stack[(*top)--];
}
```



صف

- 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 rear (انتها) نامیده می شود صورت می گیرد.
- صف یک لیست FIFO (First-In-First-Out) نیز نامیده می شود.





صف اتوبوس

صف اتوبوس

front

rear

صف اتوبوس

front

rear

صف اتوبوس

front

rear

صف ها به عنوان یک نوع داده مجرد

---

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

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

front یک محل قبل از اولین عنصر

rear محل آخرین عنصر

```
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 IsFullQ برابر true شود.
- مثال

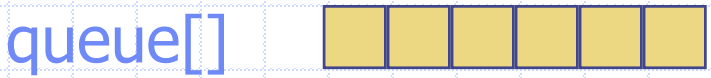
front	*rear*	*Q*[0]	*Q*[1]	*Q*[2]	*Q*[3]	Comments
−1	−1					queue is empty
−1	0	J1				Job 1 is added
−1	1	J1	J2			Job 2 is added
−1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

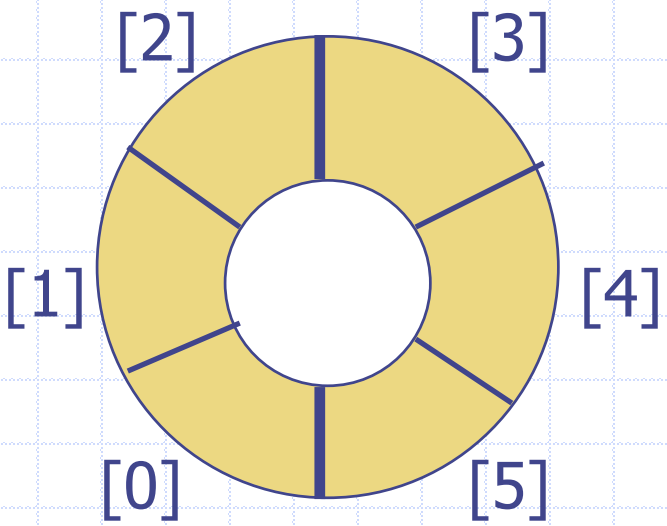
- صف به تدریج به سمت راست شیفت پیدا می کند
- در این حالت queue_full باید تمام صف را به چپ شیفت دهد به گونه ای که عنصر اول صف در مکان صفر ارایه قرار گیرد front برابر 1- و rear به صورت مناسب تصحیح شود.
- شیفت زمان گیر پیچیدگی زمانی queue_full برابر **O(MAX_QUEUE_SIZE)**

صف حلقوی

پیاده سازی ۲

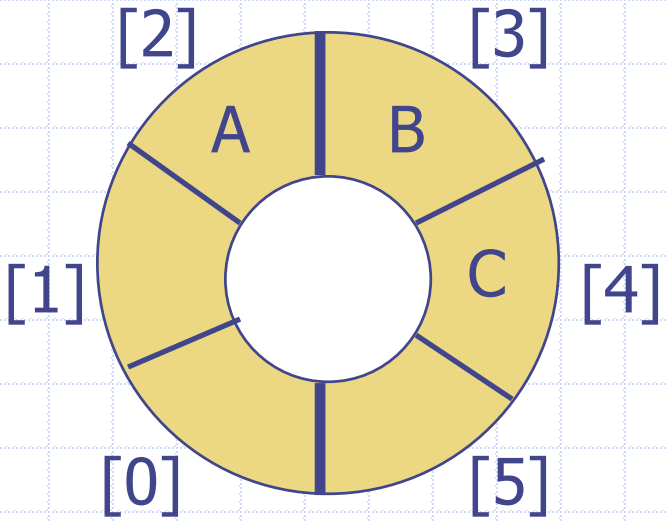
از یک ارایه 1D استفاده کرده و آن را به صورت حلقوی در نظر بگیریم





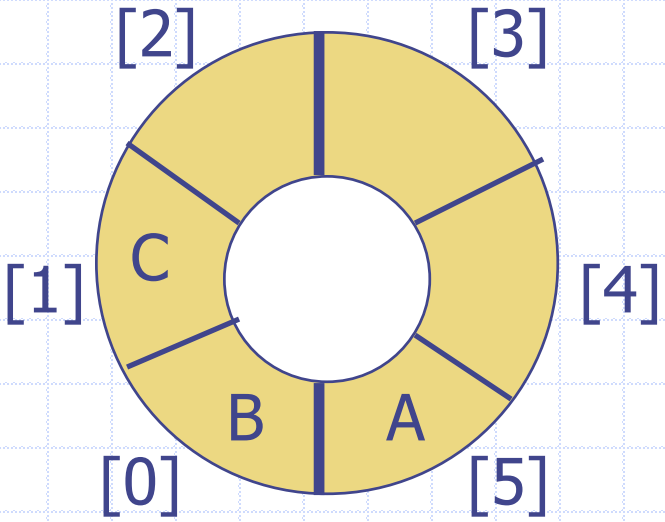
صف حلقوی

- یک صف حلقوی دلخواه با ۳ عنصر



صف حلقوی

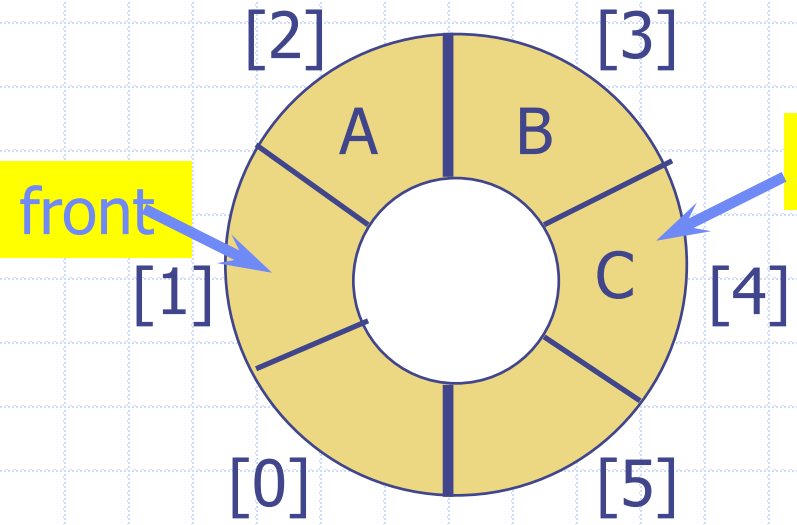
- یک صف حلقوی دلخواه دیگر با ۳ عنصر

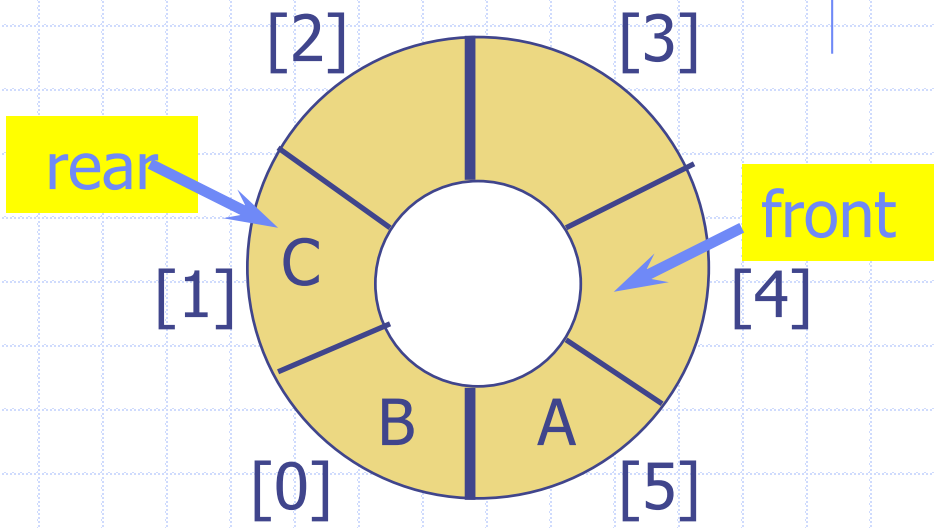


صف حلقوی

front یک محل قبل از اولین عنصر

rear محل آخرین عنصر

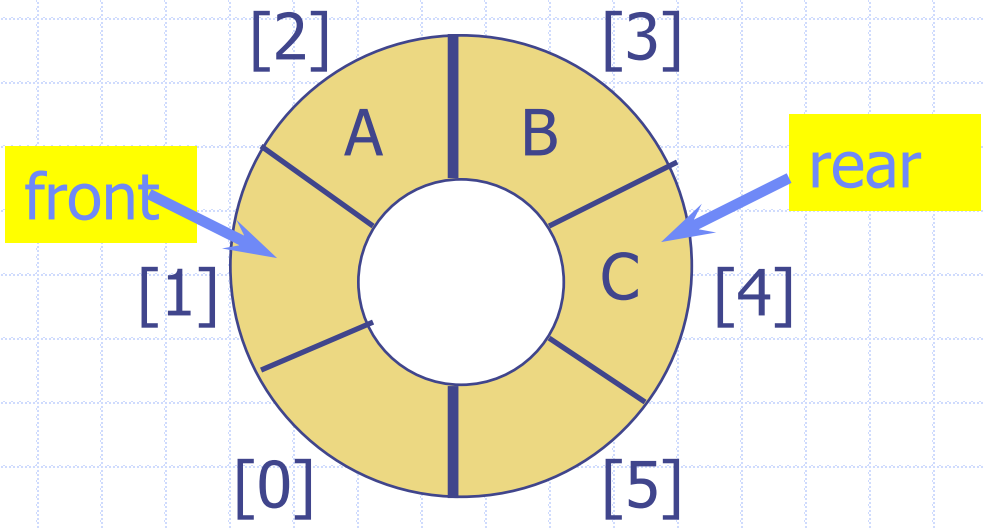




صف حلقوی

- **اضافه کردن به صف**

rear را در جهت عقربه های ساعت یک واحد افزایش دهید

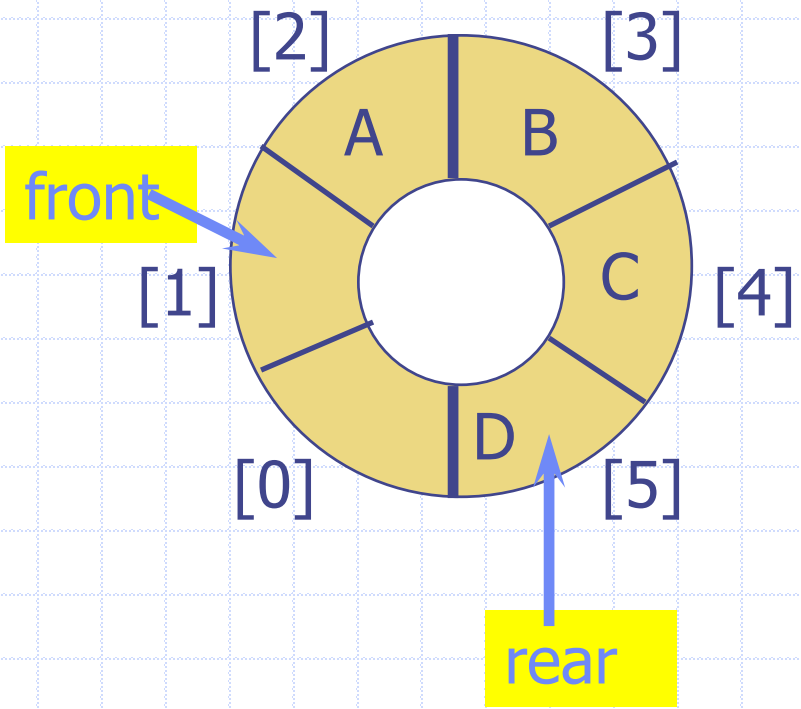


صف حلقوی

- **اضافه کردن به صف**

rear را در جهت عقربه های ساعت یک واحد افزایش دهید

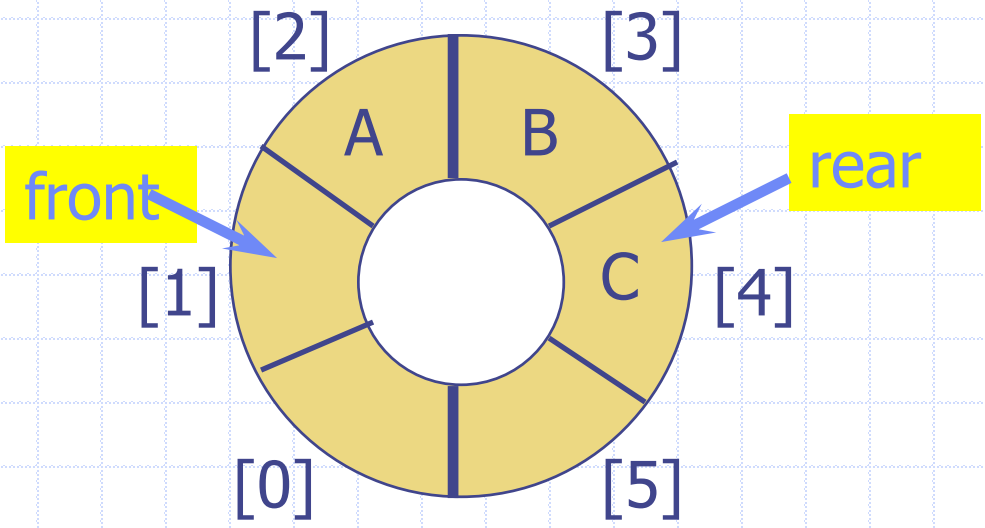
عنصر مورد نظر را در محل queue[rear] قرار دهید



صف حلقوی

- **حذف کردن یک عنصر**

front را در جهت عقربه های ساعت یک واحد افزایش دهید

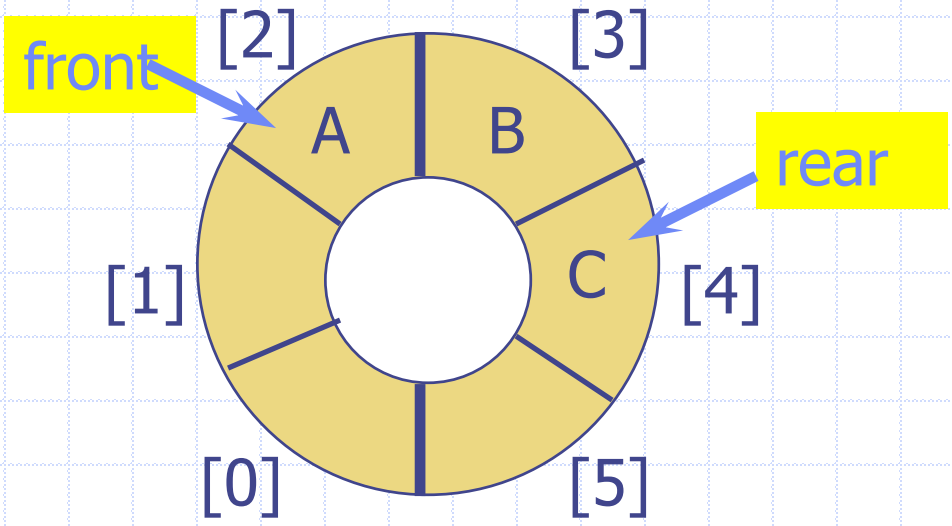


صف حلقوی

- **حذف کردن از صف**

frontرا در جهت عقربه های ساعت یک واحد افزایش دهید

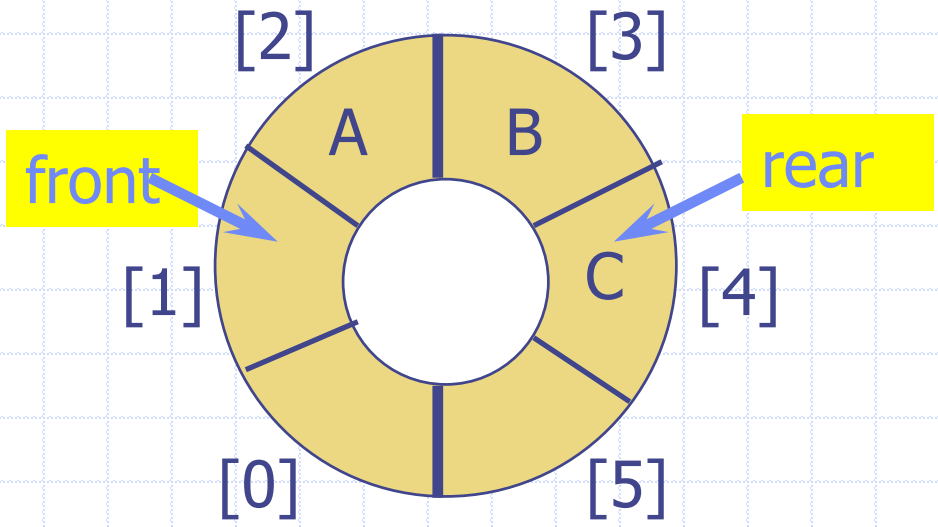
عنصر مورد نظر را از محل queue[front] استخراج کنید



صف حلقوی

- اضافه کردن به **rear** در جهت عقربه های ساعت

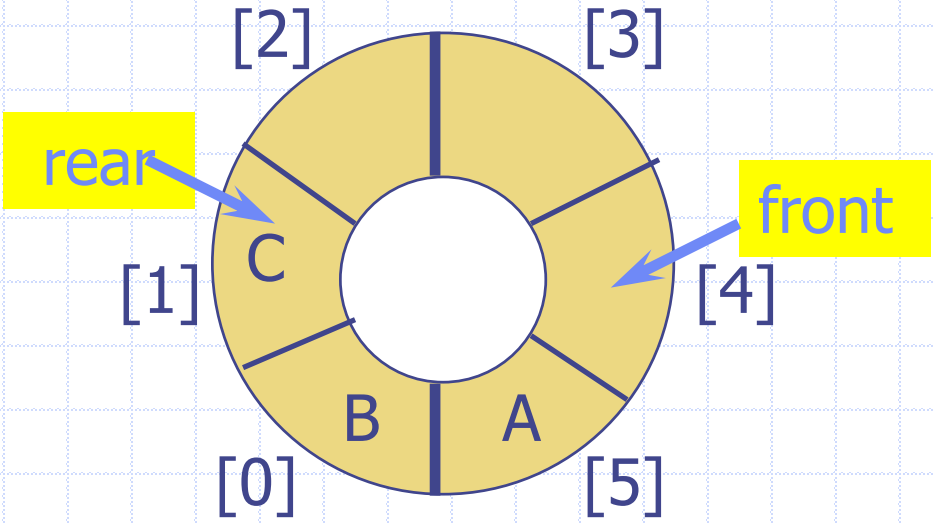
```
rear++;
if (rear = = capacity) rear = 0;
```




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

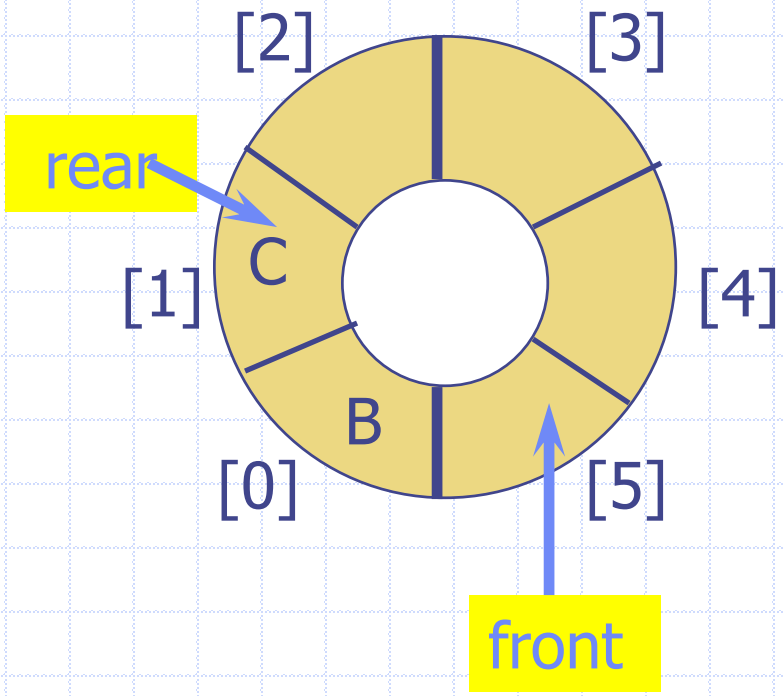
صف حلقوی

- صف خالی



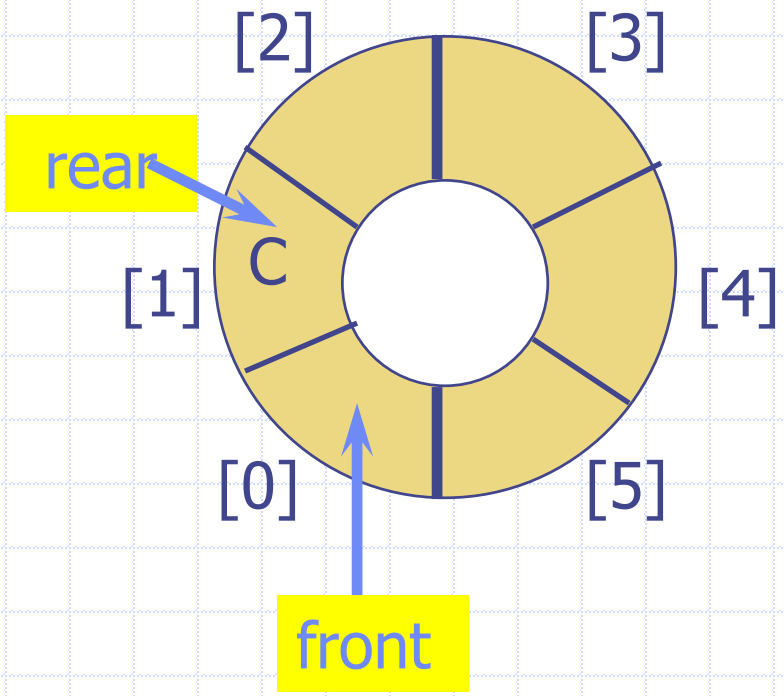
صف حلقوی

- صف خالی



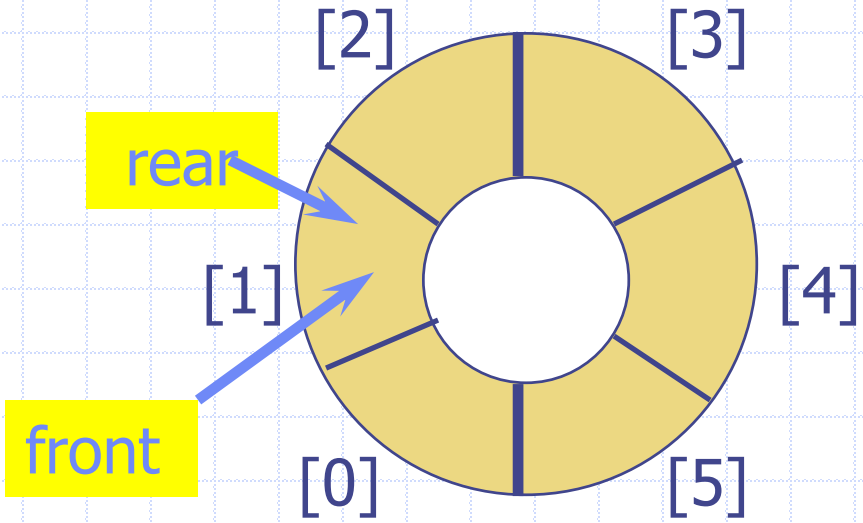
صف حلقوی

- صف خالی



صف حلقوی

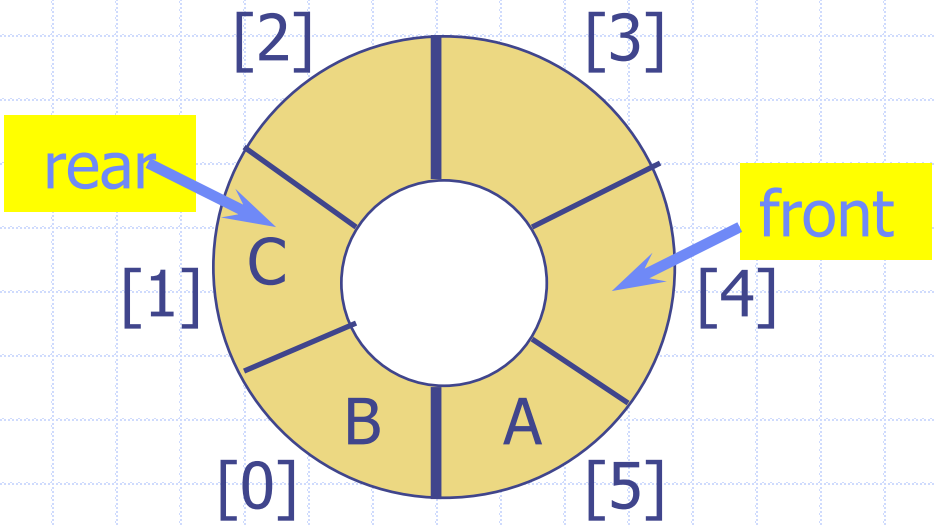
- صف خالی



- پس از چند عمل حذف صف خالی می شود، front=rear
- هنگامی که یک صف ایجاد می شود خالی است
- پس مقدار اولیه front = rear = 0

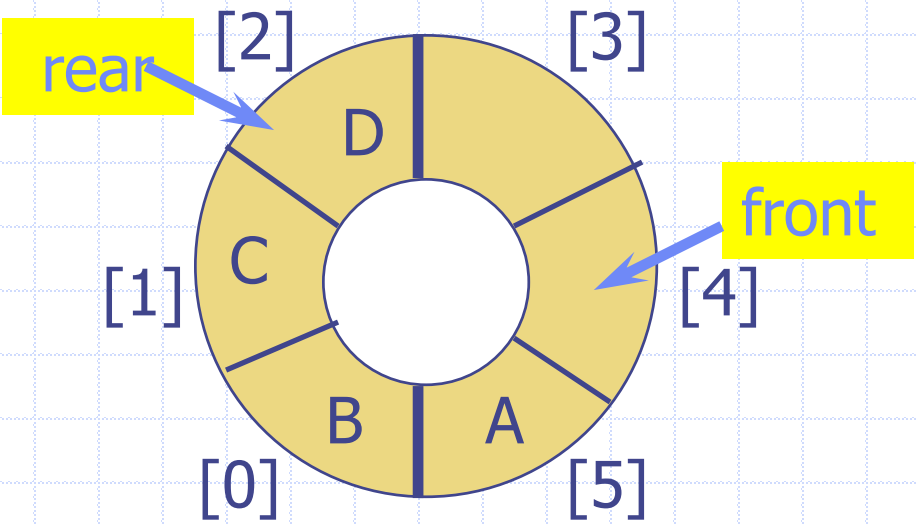
صف حلقوی

- صف پر



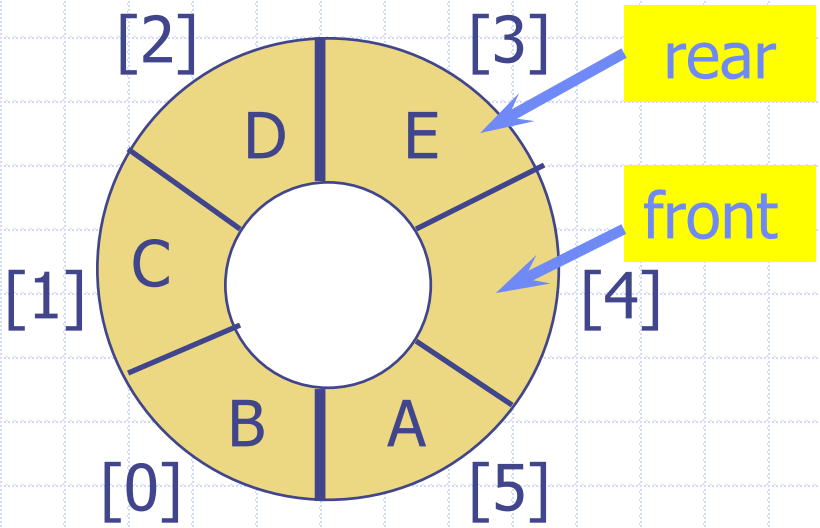
صف حلقوی

- صف پر



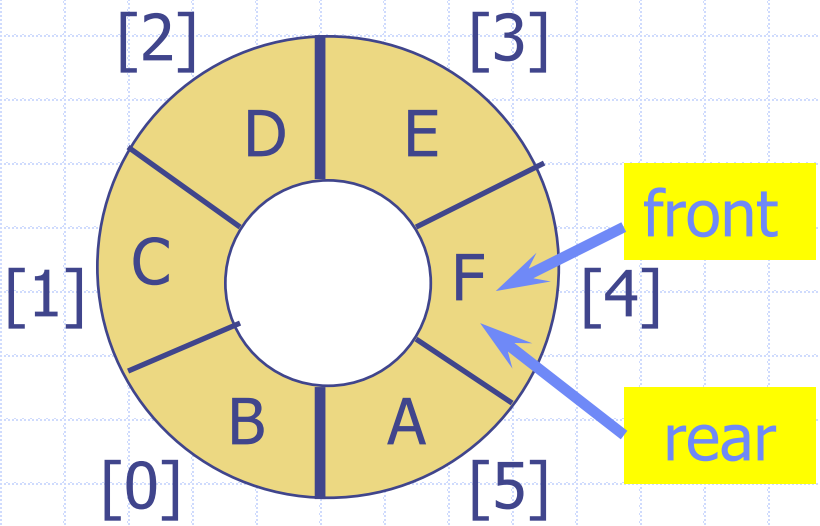
صف حلقوی

- 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 front=rear شود
- بنابراین نمی توان بین صف خالی و پر تمایز قائل شد

صف حلقوی

- **تمایز بین صف پر و خالی**
  - اجازه ندهیم صف به طور کامل پر شود
    - اگر اضافه کردن یک عنصر باعث پر شدن صف می شود طول ارایه را اضافه کنیم یا این عنصر را اضافه نکنیم.
  - یک متغیر بولی lastOperationIsPush تعریف کنیم
    - پس از هر اضافه کردن این متغیر true می شود.
    - پس از هر حذف کردن این متغیر false می شود.

Queue is empty *iff* (front == rear) && !lastOperationIsPush

Queue is full *iff* (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 iff (size == 0)

Queue is full iff (size == arrayLength)

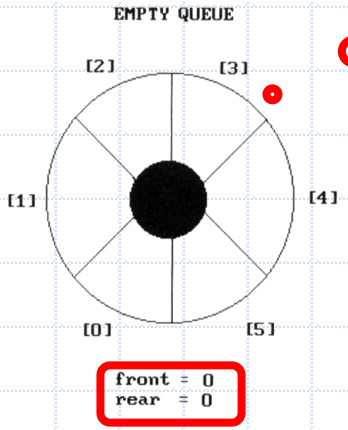
کارایی هنگامی که از راهکار اول استفاده شود بیشتر است

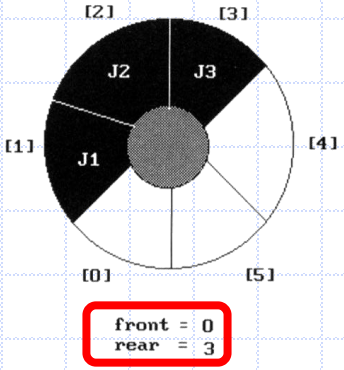
صف حلقوی

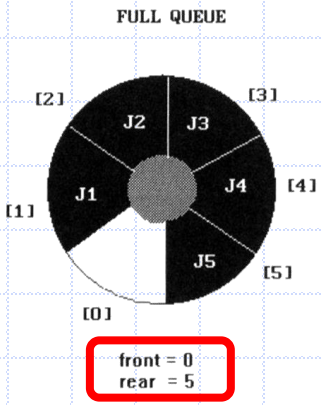
صف خالی
front=rear

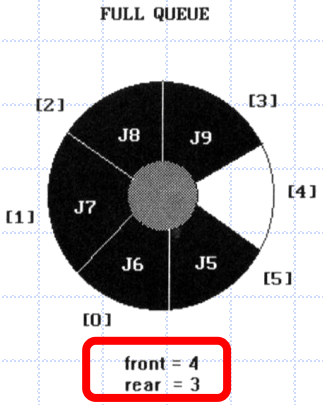
صف غیر خالی

صف پر
front=(rear+1)%maxzize









مشکل: یک فضای خالی باقی می ماند

صف حلقوی

```
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];
}
```

تطبیق پرانتزهای یک عبارت

- به گونه ای زوجهای (u,v) را بیابید 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)

  (8,12)

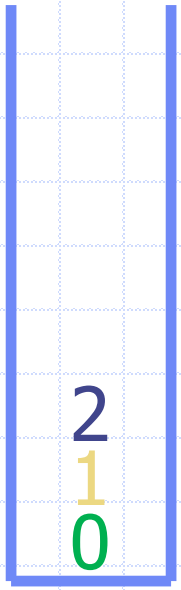
- پرانتز راست موقعیت ۵ پرانتز چپ متناظر ندارد
- پرانتز چپ موقعیت ۷ پرانتز راست متناظر ندارد

تطبیق پرانتزهای یک عبارت

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

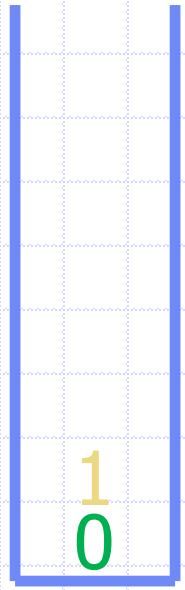
مثال

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



مثال

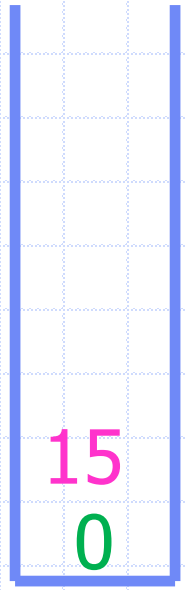
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6)

مثال

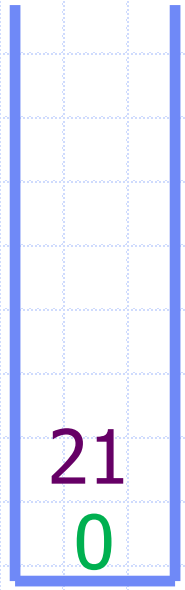
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13)

مثال

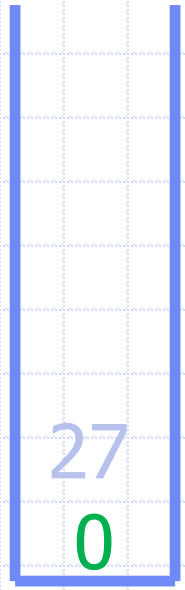
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13)(15,19)

مثال

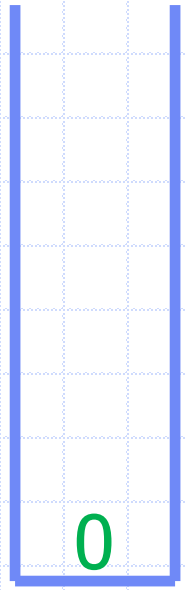
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13)(15,19)(21,25)

مثال

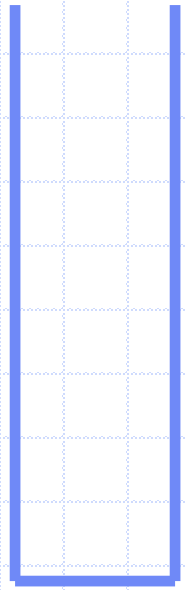
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



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

مثال

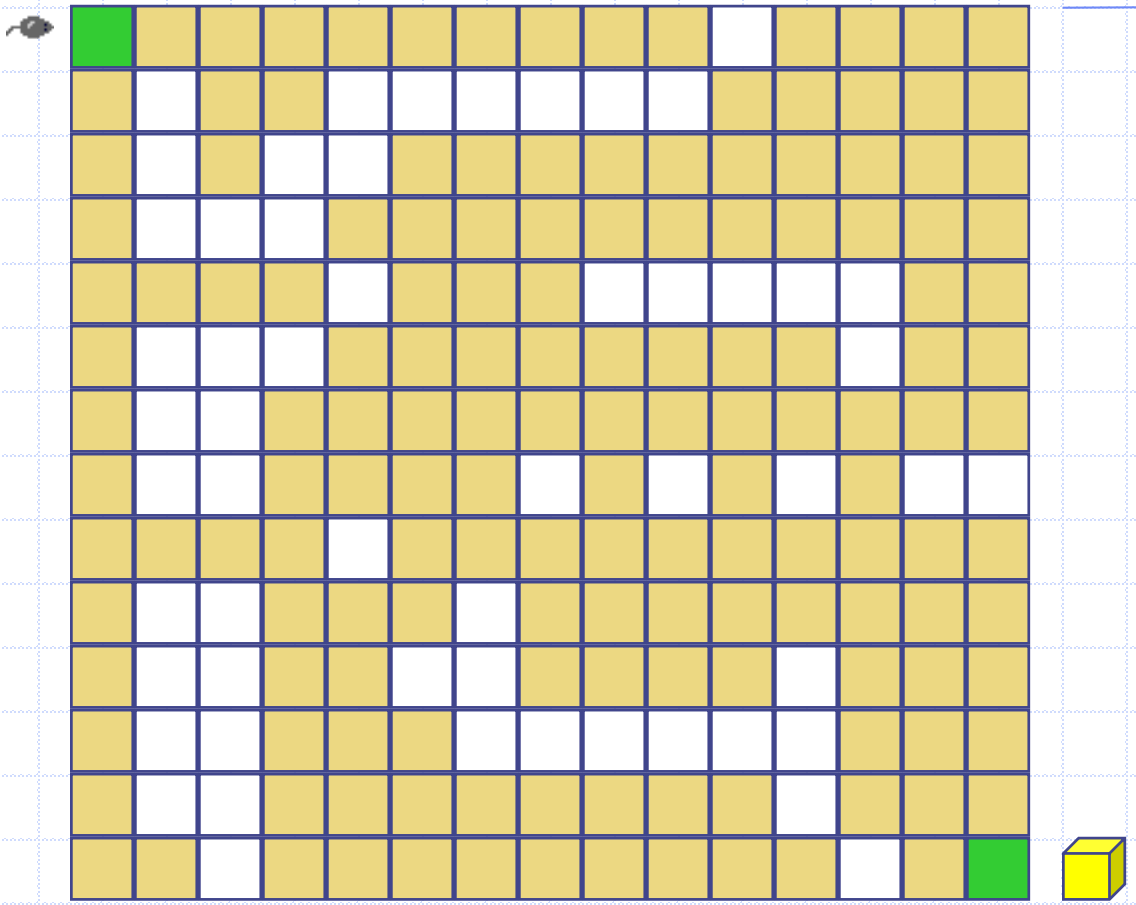
- (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



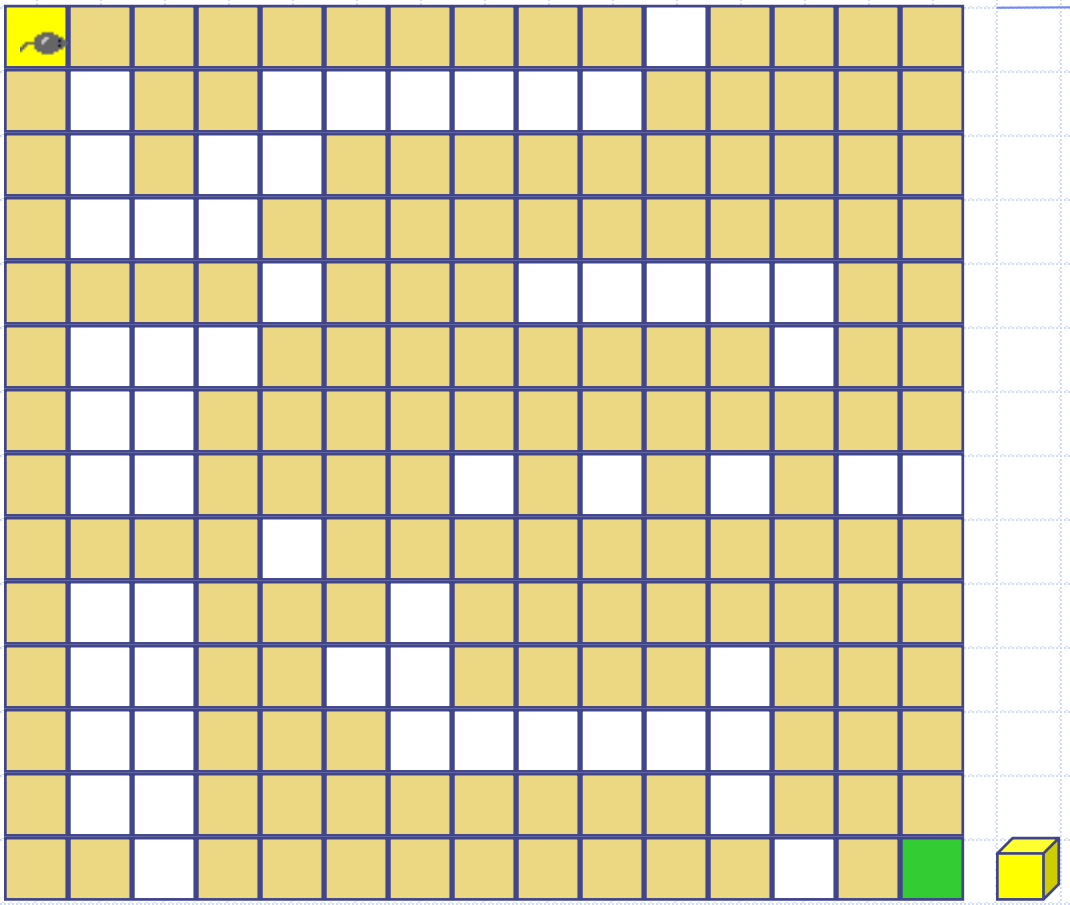
(2,6) (1,13)(15,19)(21,25) (27,31) (0,32)

and so on

بازی Maze

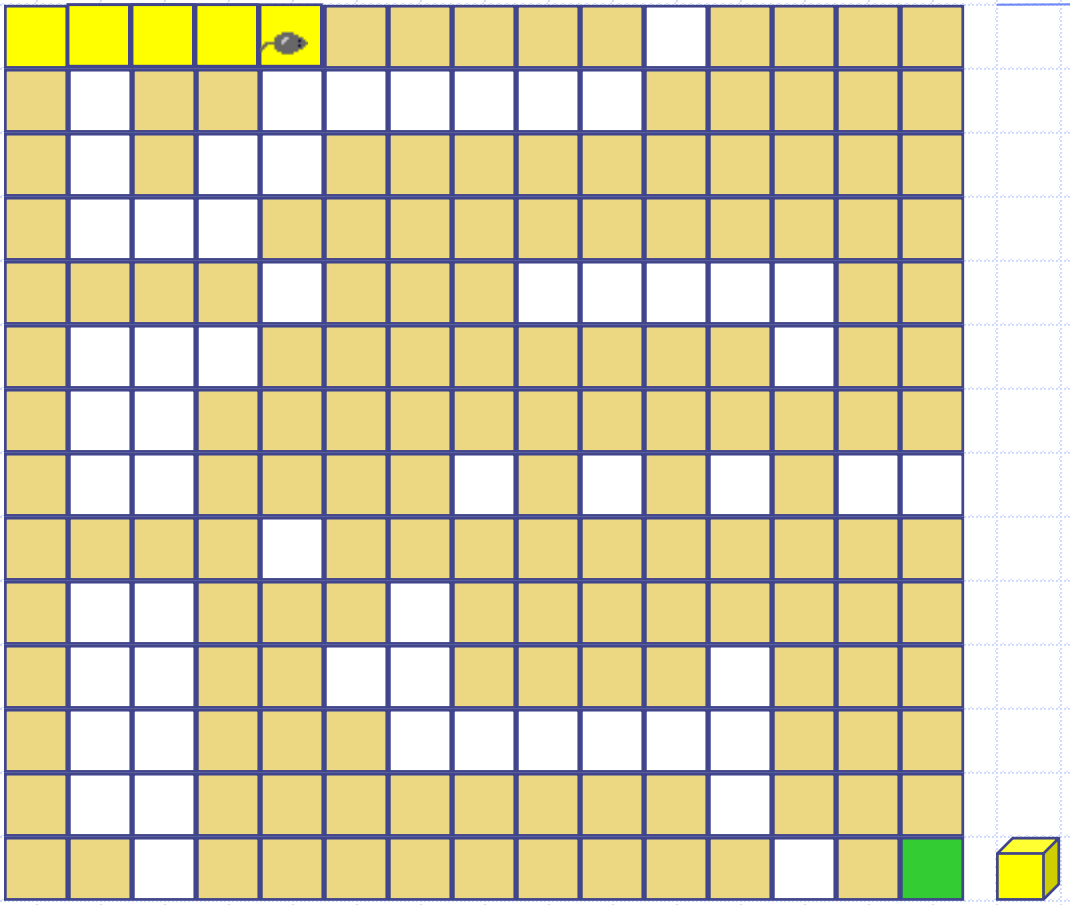


بازی Maze

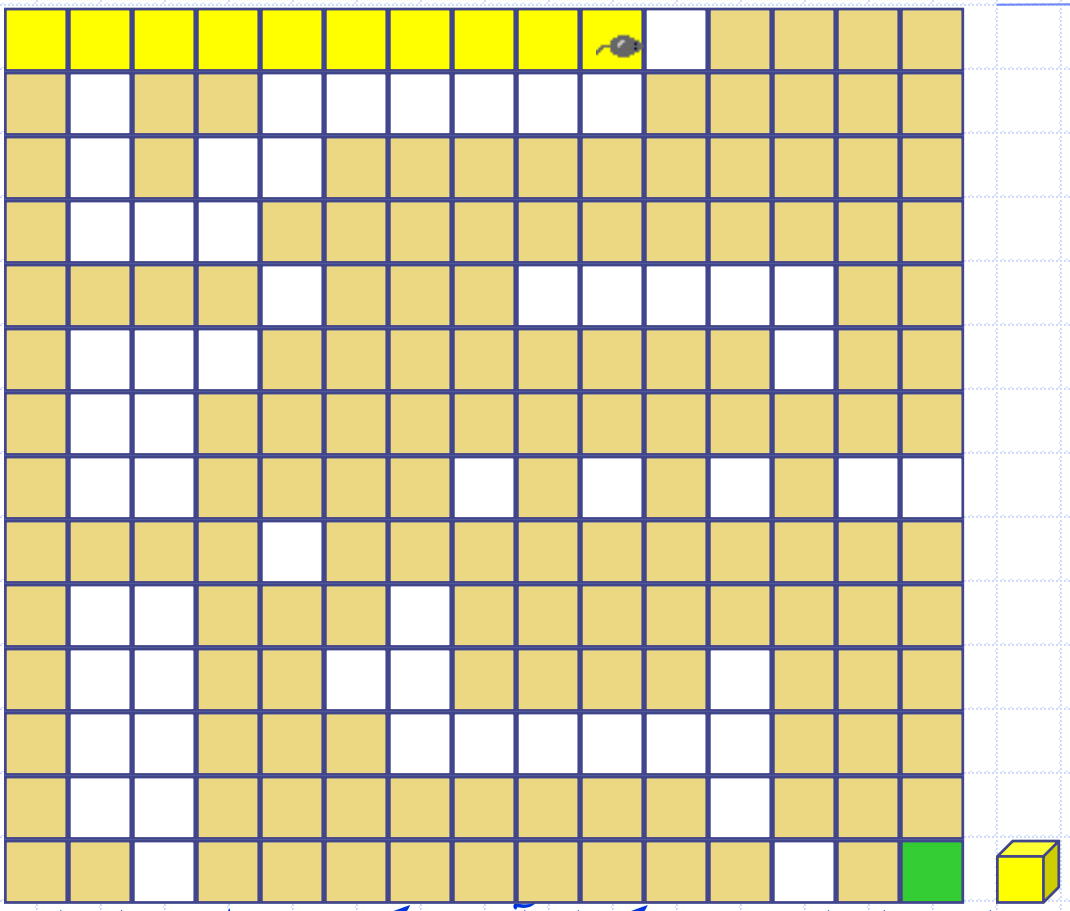


- حرکات مجاز: راست، پایین، چپ، بالا
- خانه ها را علامت گذاری کنید تا از بازدید مجدد جلوگیری شود.

Maze بازي

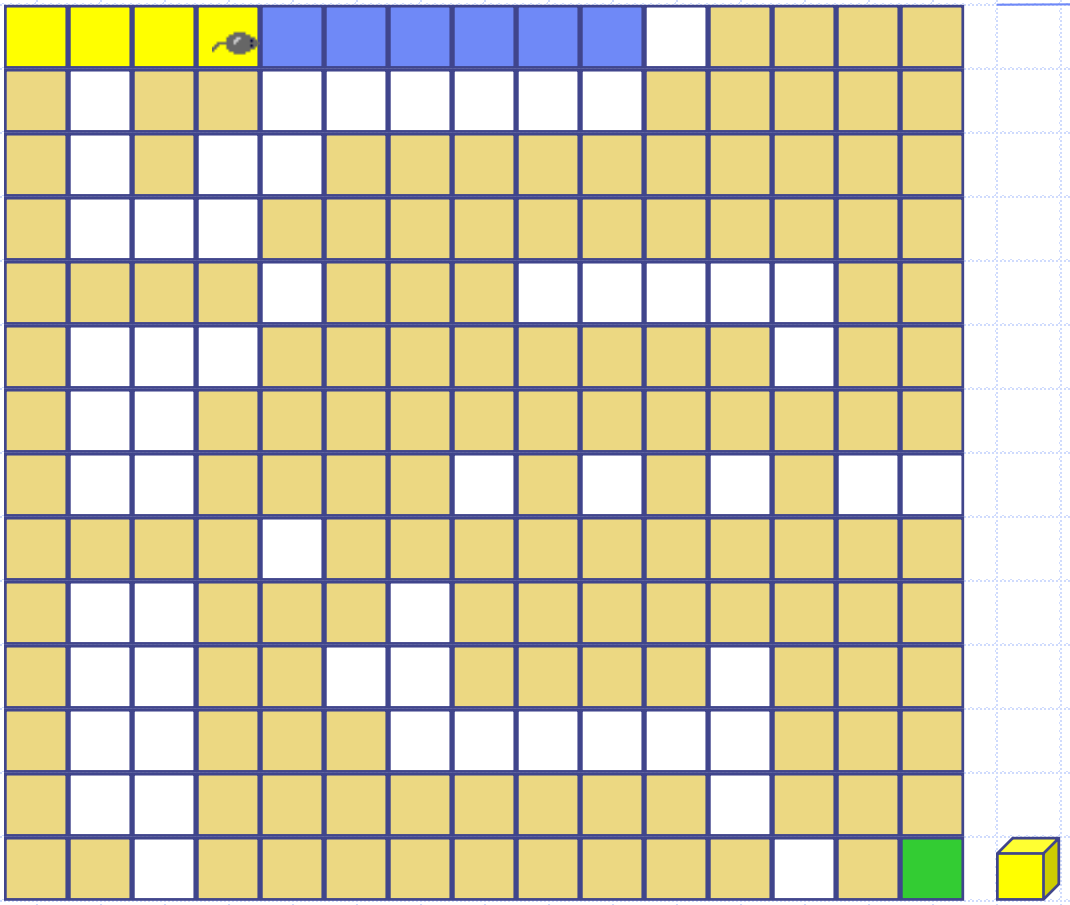


بازی Maze



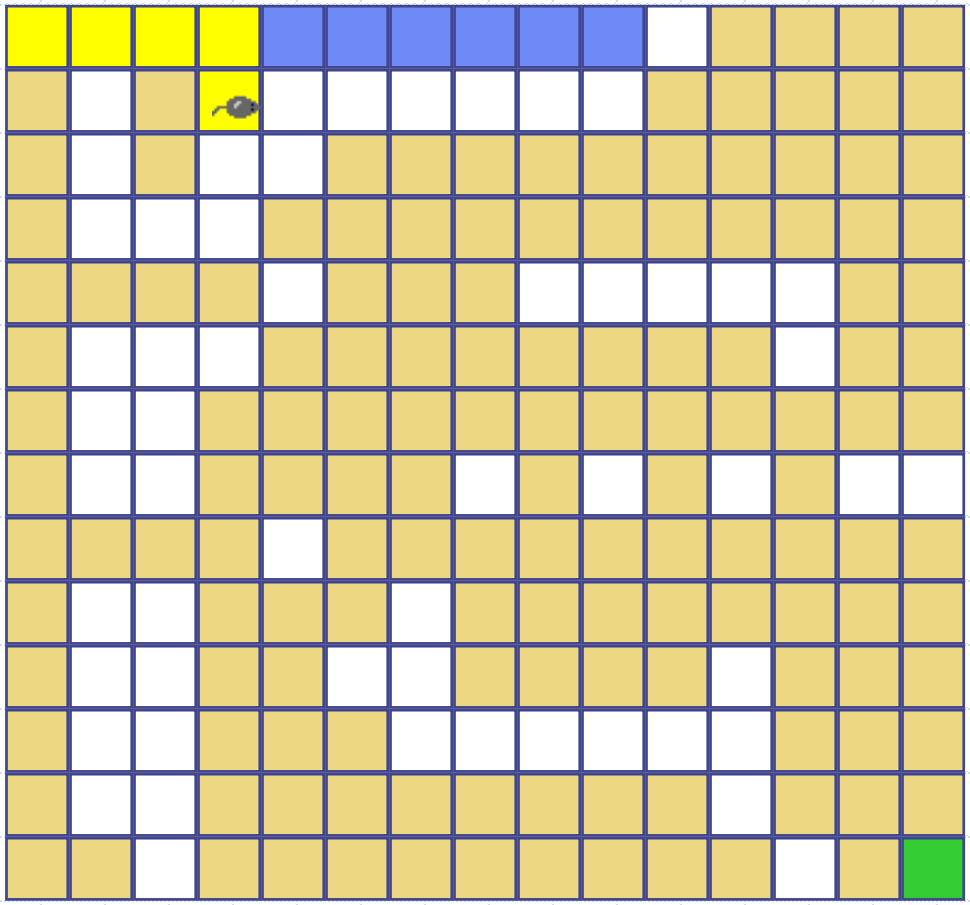
- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد

بازی Maze



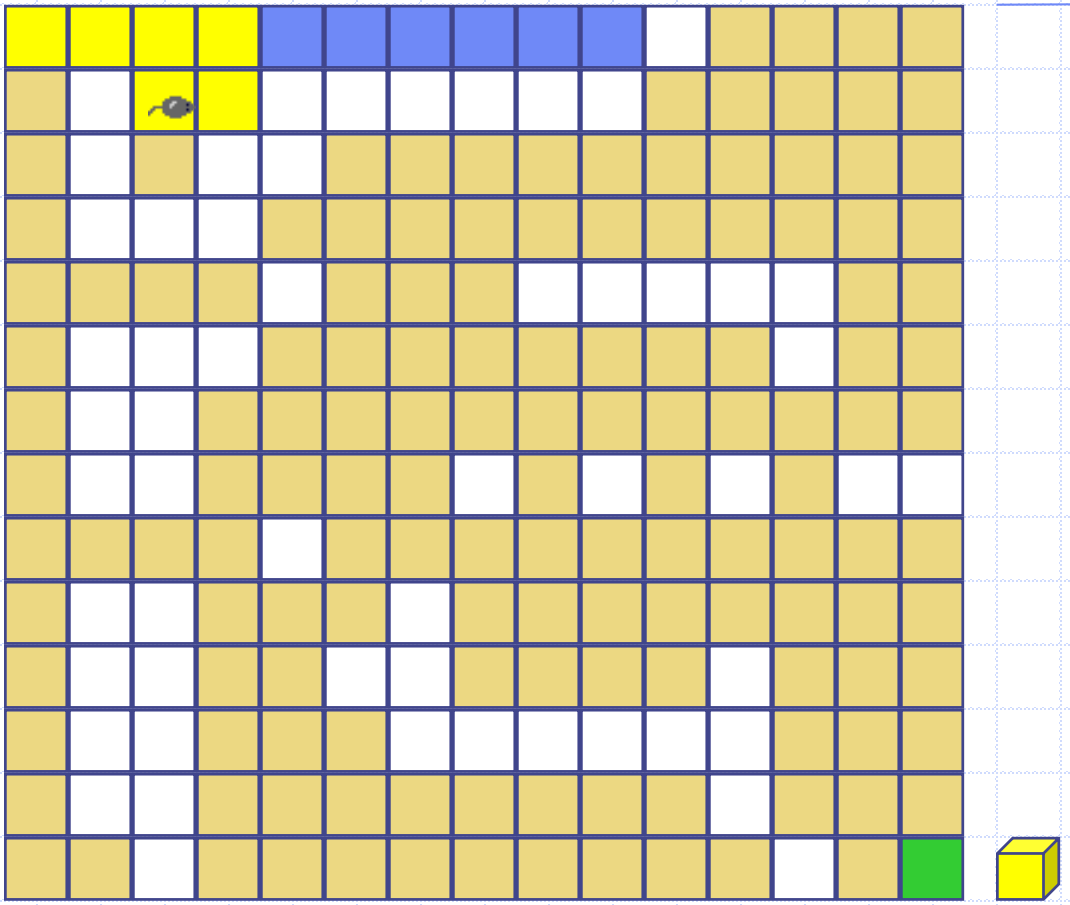
- به سمت پایین حرکت کنید.

بازی Maze



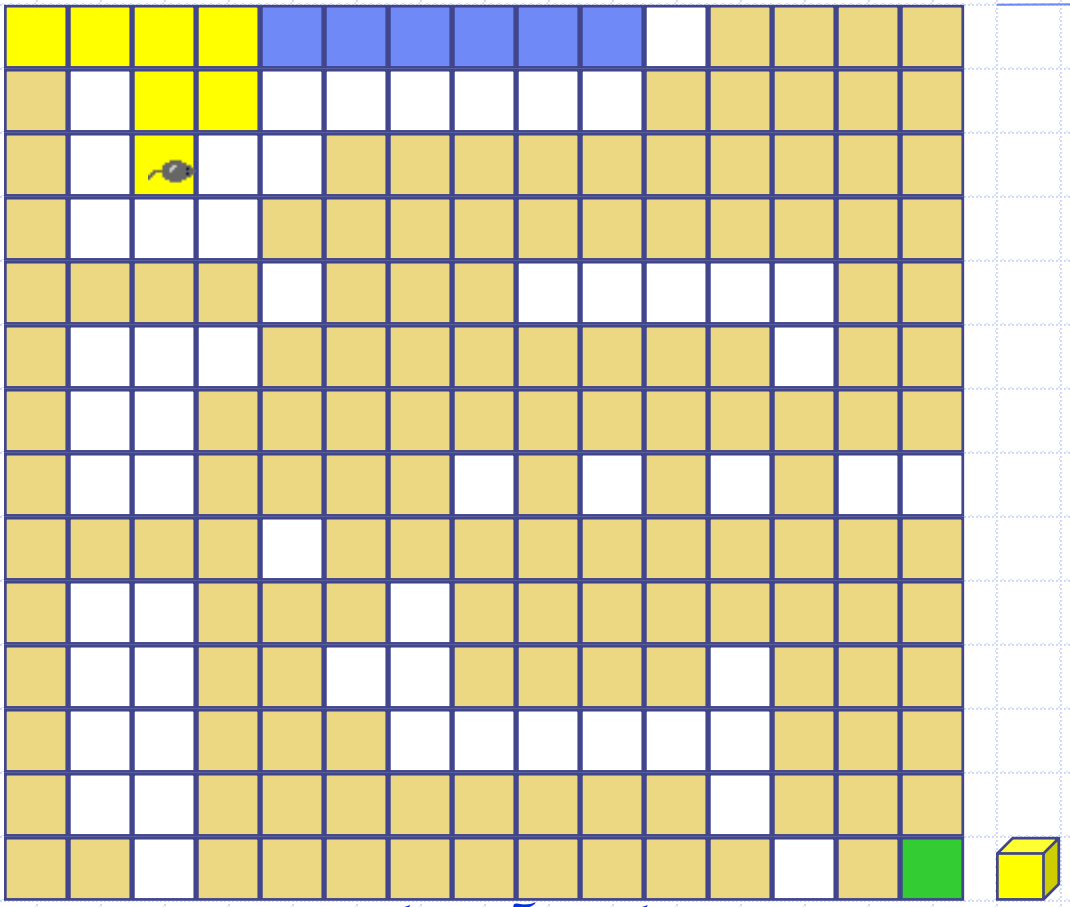
- به سمت چپ حرکت کنید.

بازی Maze



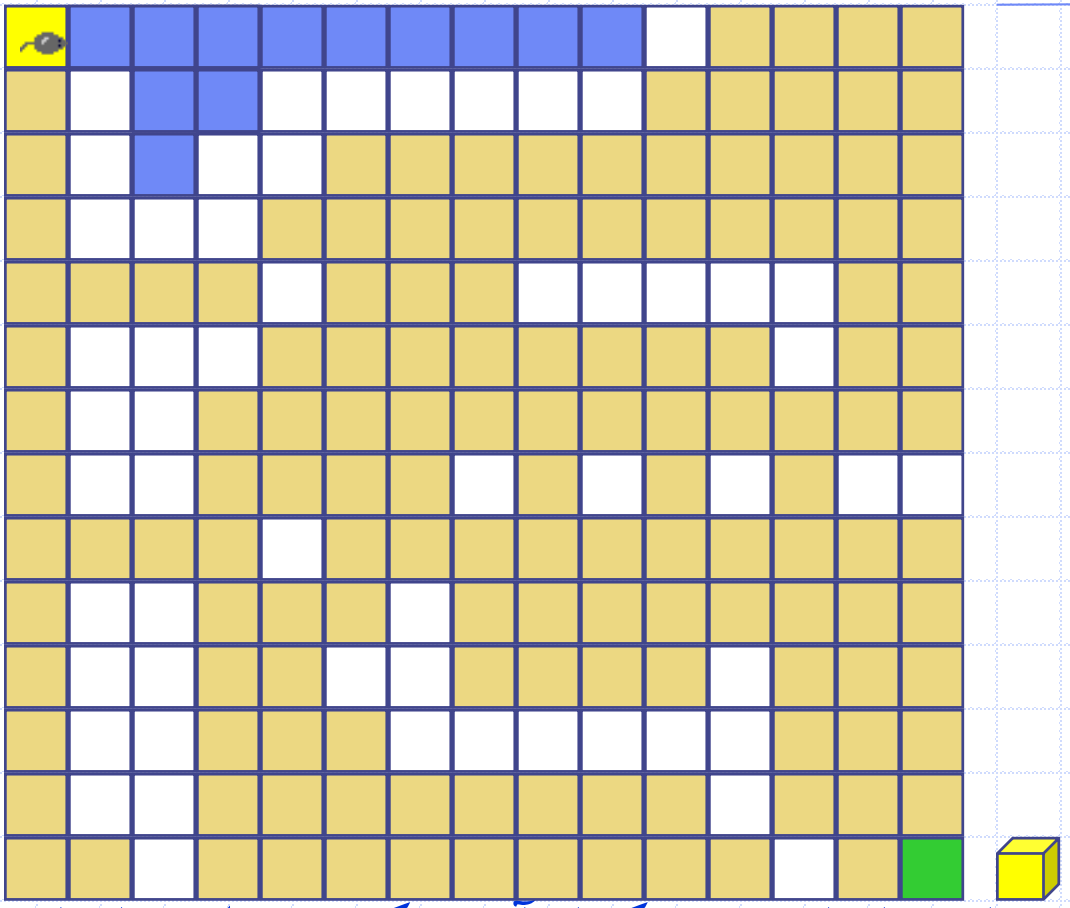
- به سمت پایین حرکت کنید.

بازی Maze



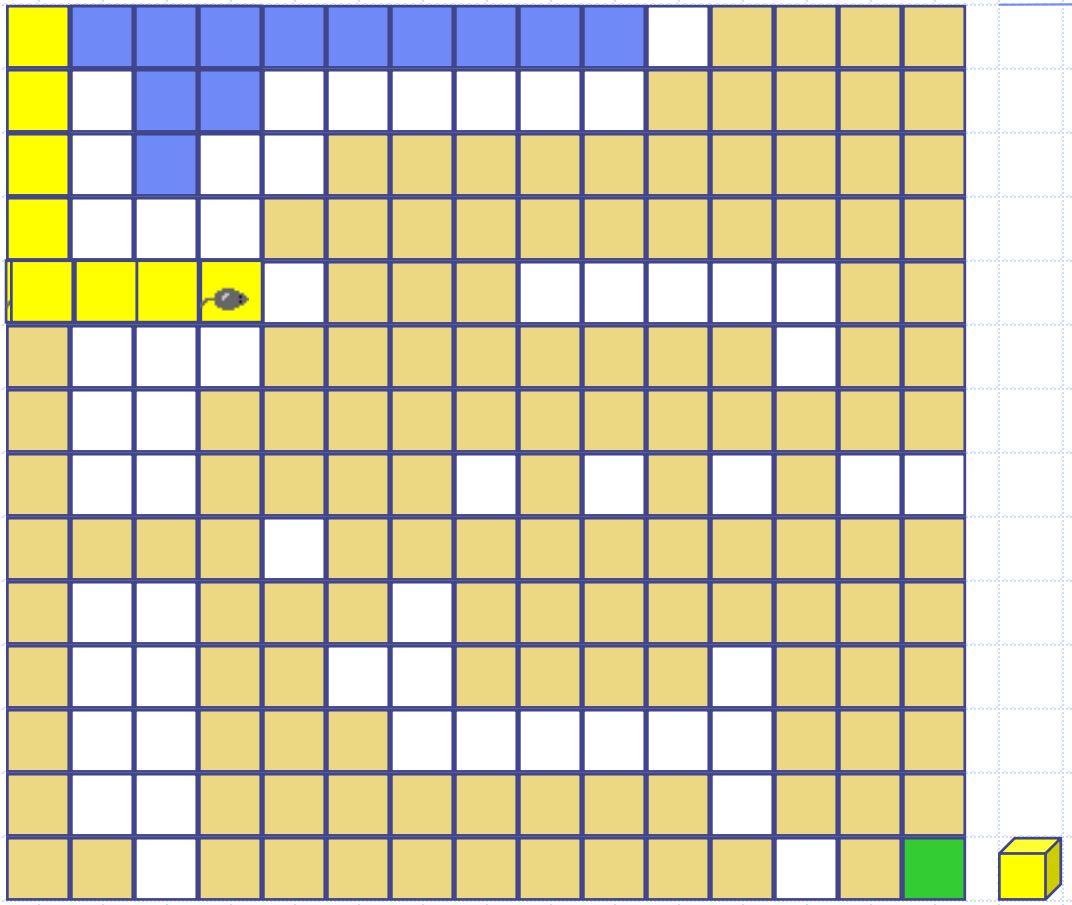
- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد

بازی Maze



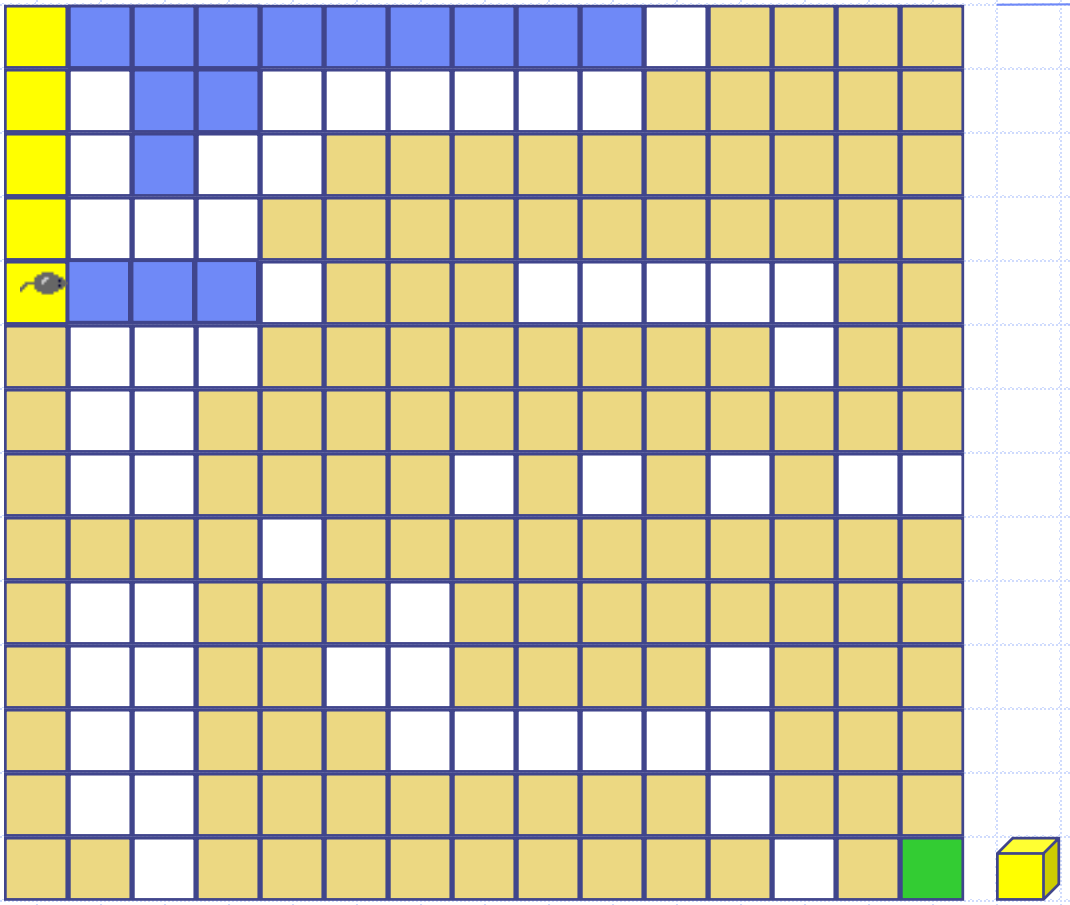
- به عقب برگردید تا به خانه ای برسید که از آن حرکت به جلو مجاز باشد.
- به سمت پایین حرکت کنید.

Rat In A Maze



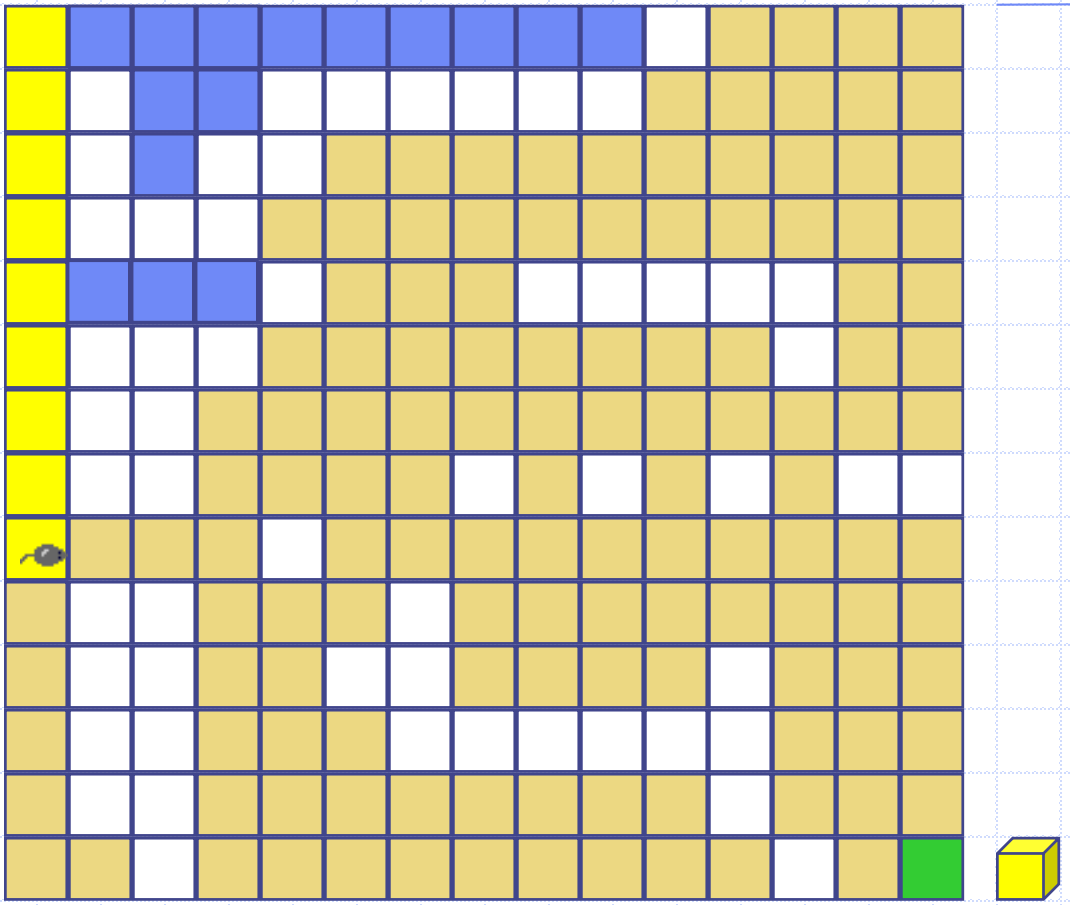
- به سمت راست حرکت کنید.
- عقبگرد.

بازی Maze



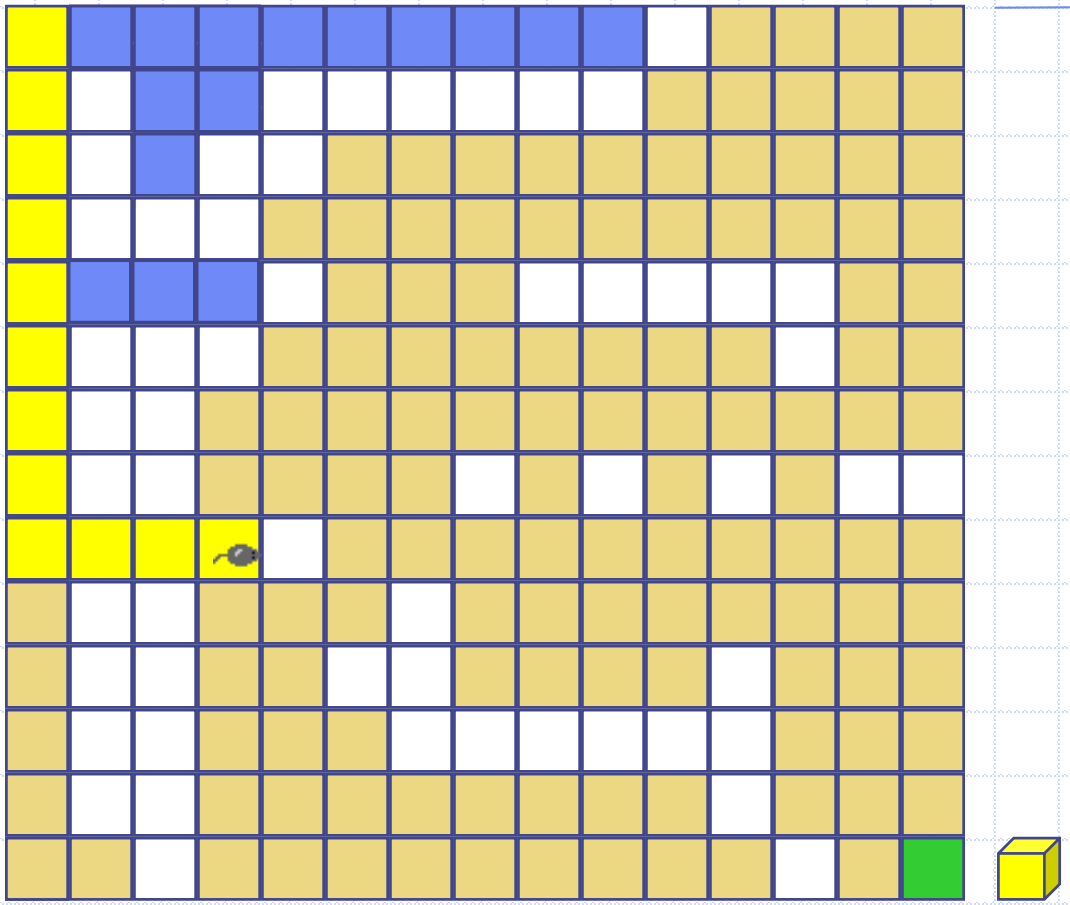
- به سمت پایین حرکت کنید.

بازی Maze



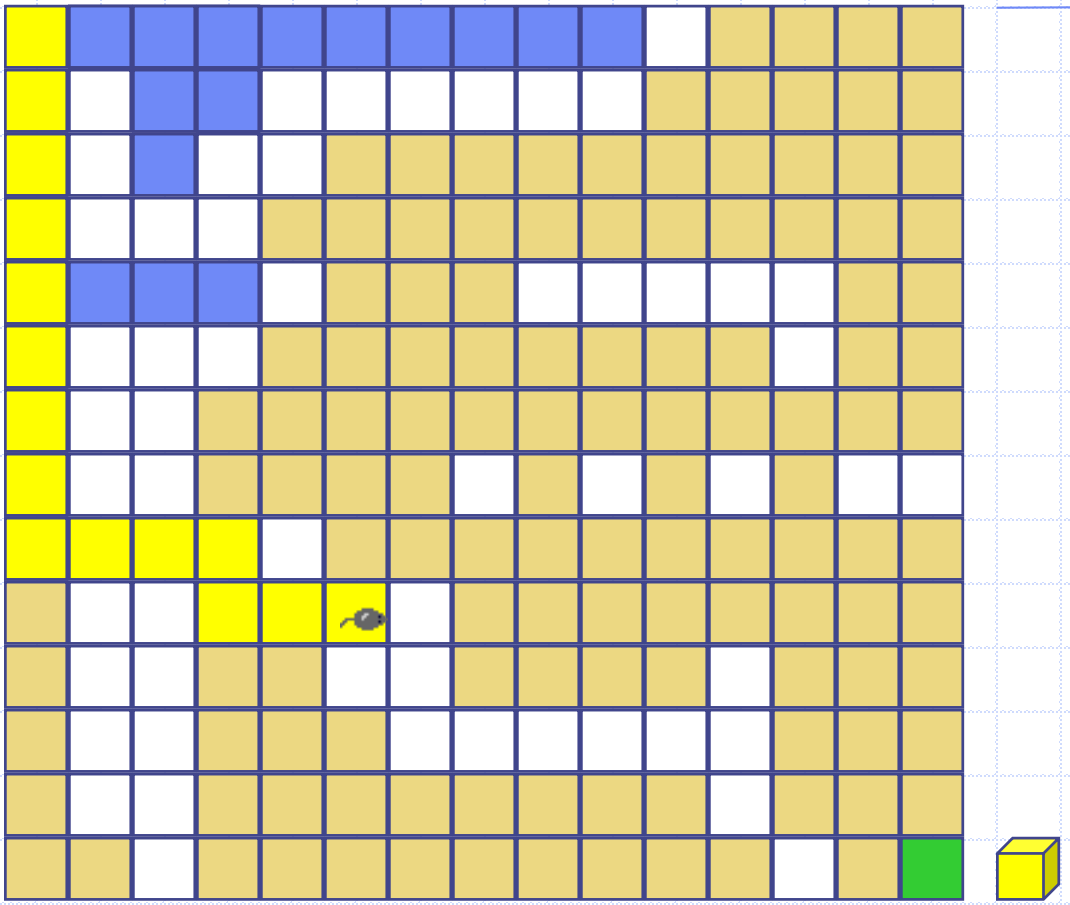
- به سمت راست حرکت کنید.

بازی Maze



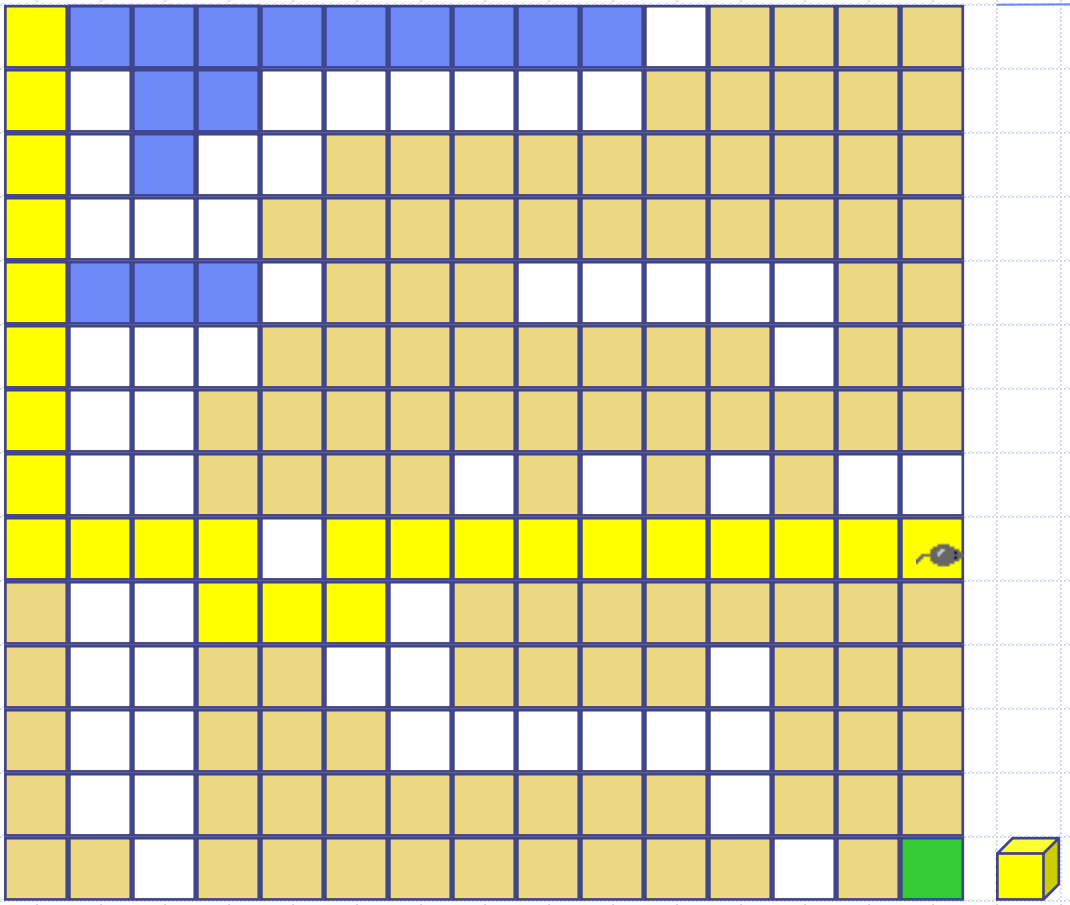
- یکی به سمت پایین و سپس به راست حرکت کنید.

بازی Maze



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

بازی Maze



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

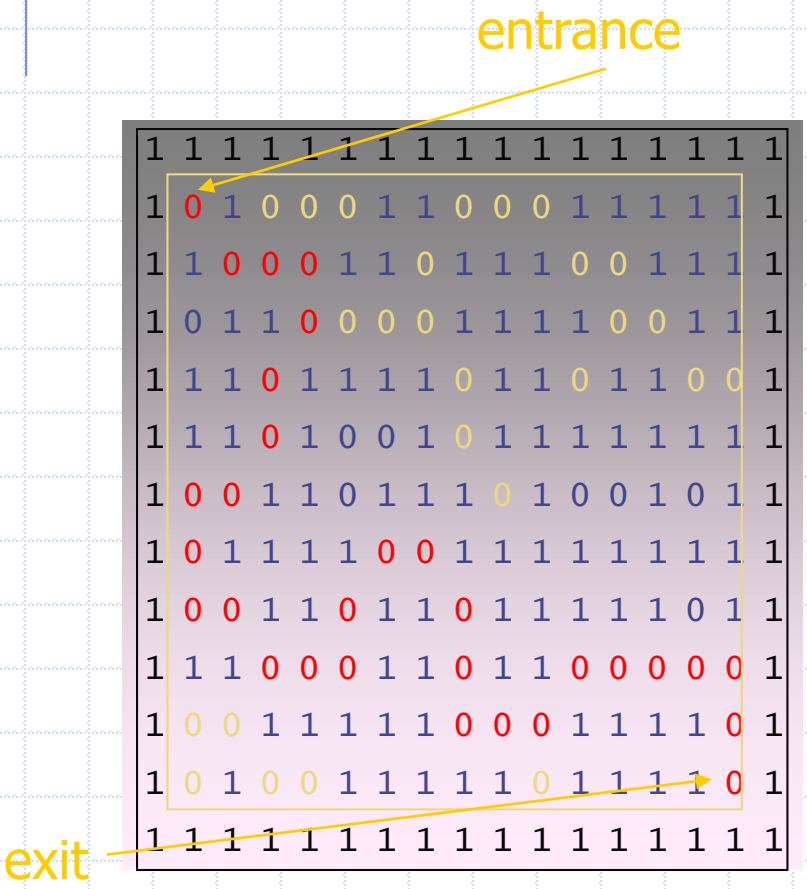
بازی Maze

- ساده ترین بازنمایی
  - استفاده از یک ارایه دوبعدی

صفر ها مسیر باز و یک ها موانع

  - همه ی نقاط ۸ همسایه ندارند

برای جلوگیری از چک کردن یک ردیف ۱ دور صفحه فرض می کنیم



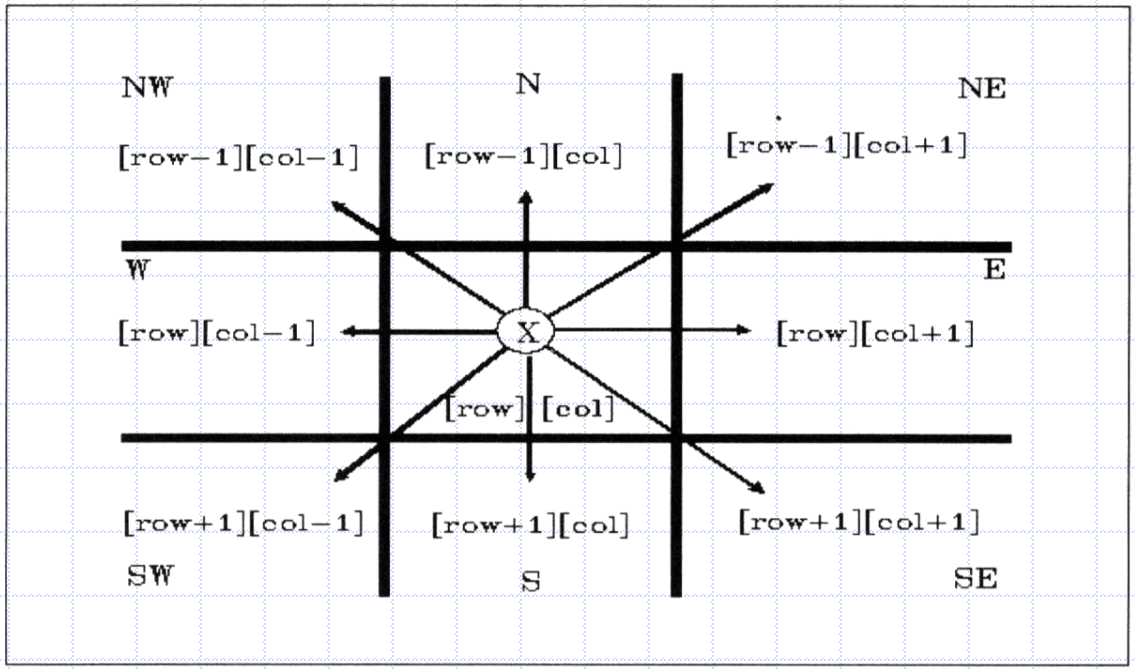
m×p maze ⇒ *(m+2) × (p+2) array*

entrance ⇒ *position [1][1]*

exit ⇒ *position[m][p]*

بازی Maze

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



بازی Maze

- پیاده سازی حرکتهای مجاز

Name	Dir	*move[dir].vert*	*move[dir].horiz*
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

```
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;
```

بازی Maze

- از یک ارایه دوبعدی mark برای نگهداری محلهای مشاهده شده تا کنون استفاده می شود.
- یک پشته برای نگهداری مسیر به کار می رود

```
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");
```

```
#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");
}
```

Name	Dir	move[dir].vert	move[dir].horiz
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

R: row
C: col
D: dir

R 1 C 1 D 1

R3 C12 D 5

R3 C13 D 6

Pop out

R2 C12 D 3

R2 C11 D 2

R1 C10 D 3

R 1 C 9 D 2

R 1 C 8 D 2

R 2 C 7 D 1

R 3 C 6 D 1

R 3 C 5 D 2

R 2 C 4 D 3

R 1 C 5 D 5

R 1 C 4 D 2

R 1 C 3 D 2

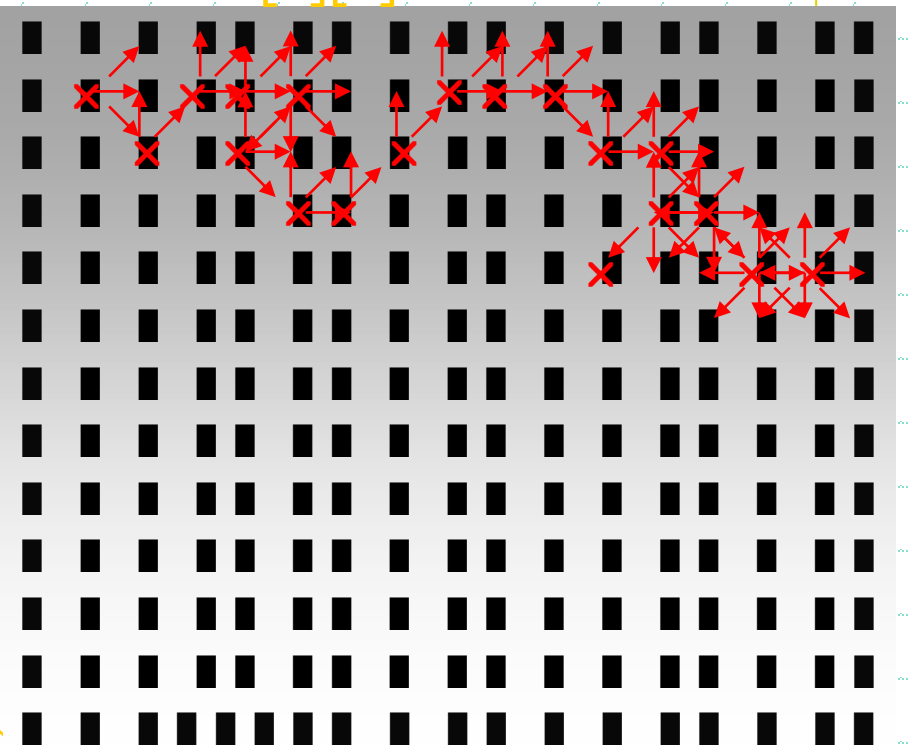
R 2 C 2 D 1

R 1 C 1 D 3

maze[1][1]: entrance

Initially set
mark[1][1]=1

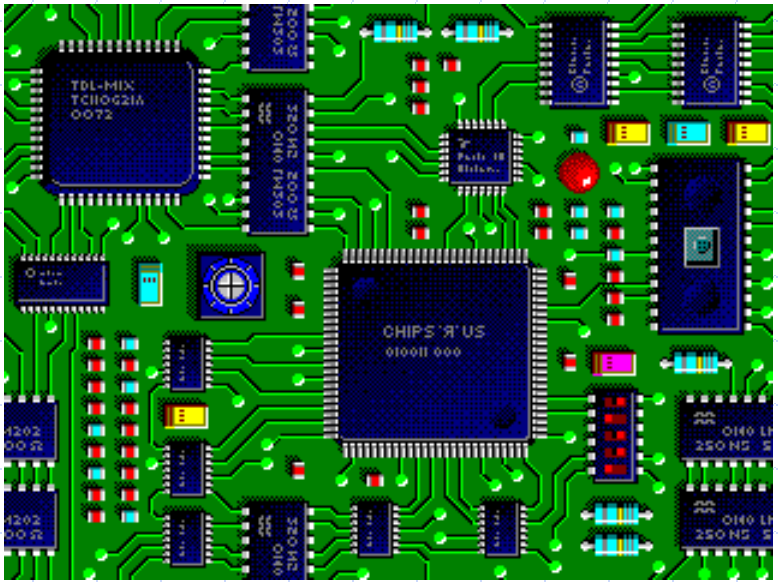
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	0	0	1	1	0	0	0	1	1	1	1	1
1	1	0	0	0	1	1	0	1	1	1	0	0	1	1	1
1	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1
1	1	1	0	1	1	1	1	0	1	1	0	1	1	0	0
1	1	1	0	1	0	0	1	0	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	1	0	1	0	0	1	0	1
1	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	0	1	1	1	1	1	0	1
1	1	1	0	0	0	1	1	0	1	1	0	0	0	0	0
1	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0
1	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1



maze[15][11]: exit

مسیریابی برای سیم

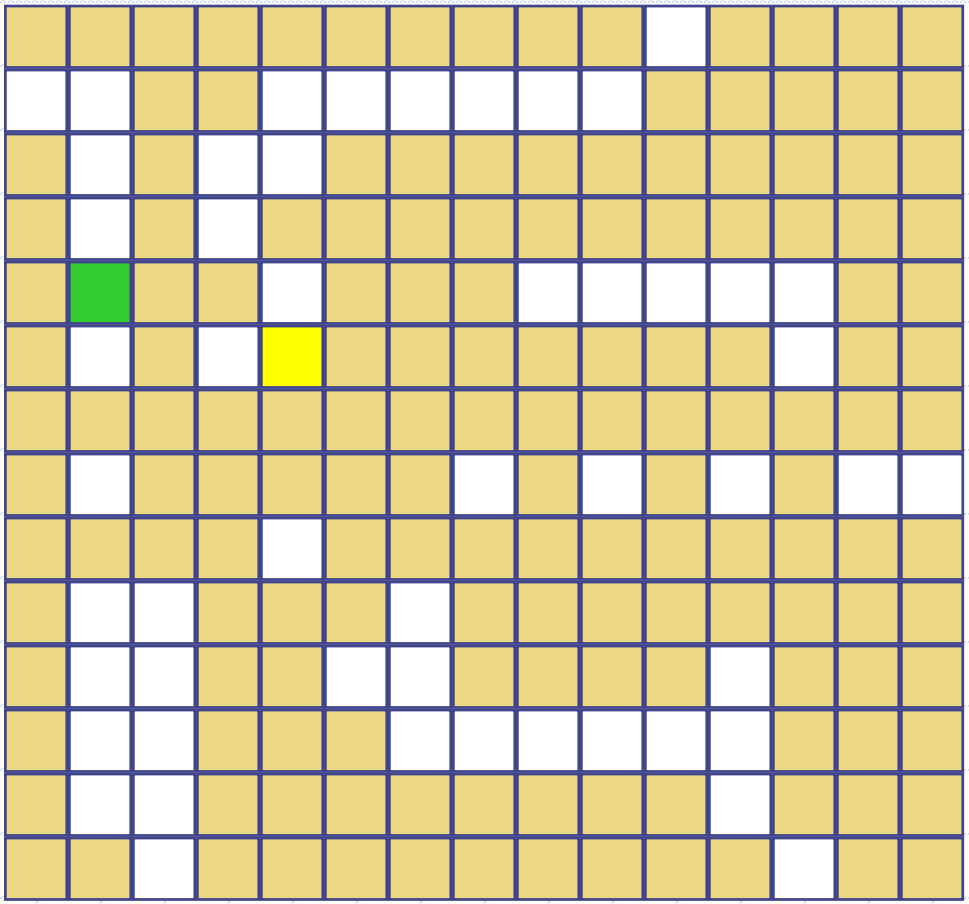
- می توان به جای پشته از صف استفاده کرد
- منجر به یافتن کوتاهترین مسیر به خروجی می شود.



مسیریابی برای سیم

start pin

end pin

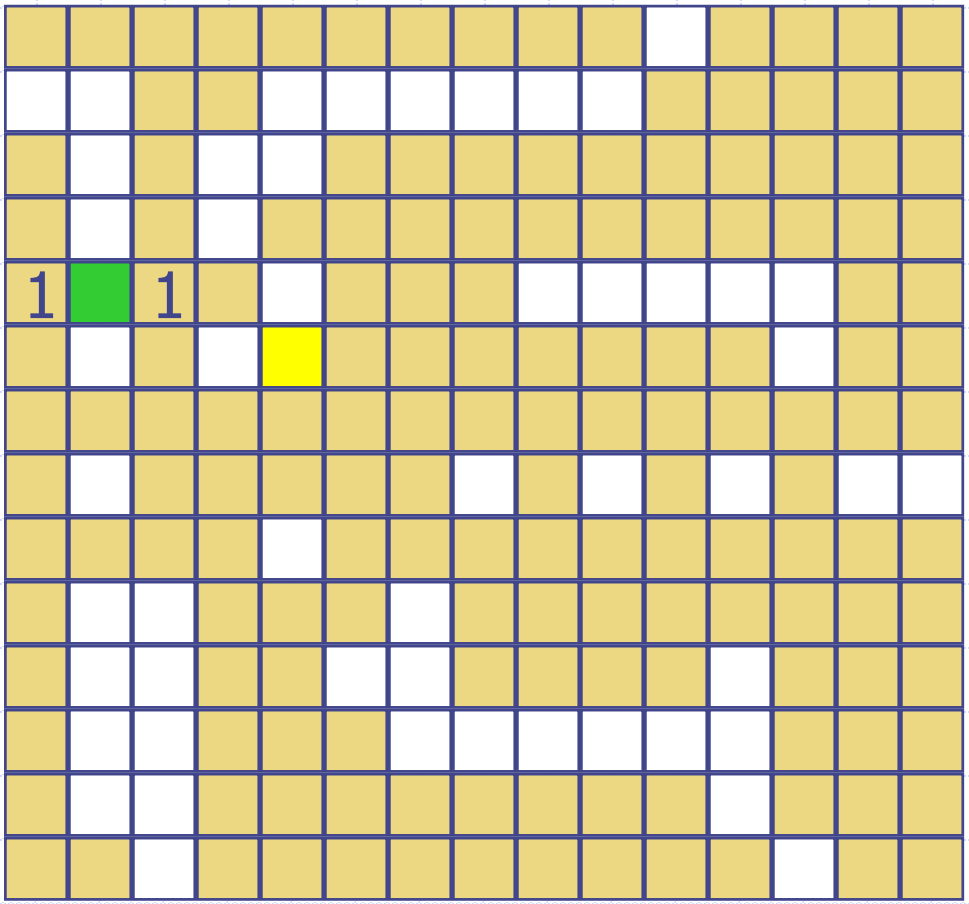


- همه مربع هایی که با فاصله یک واحد از start قابل دسترس هستند را برچسب بزنید

مسیریابی برای سیم

start pin

end pin

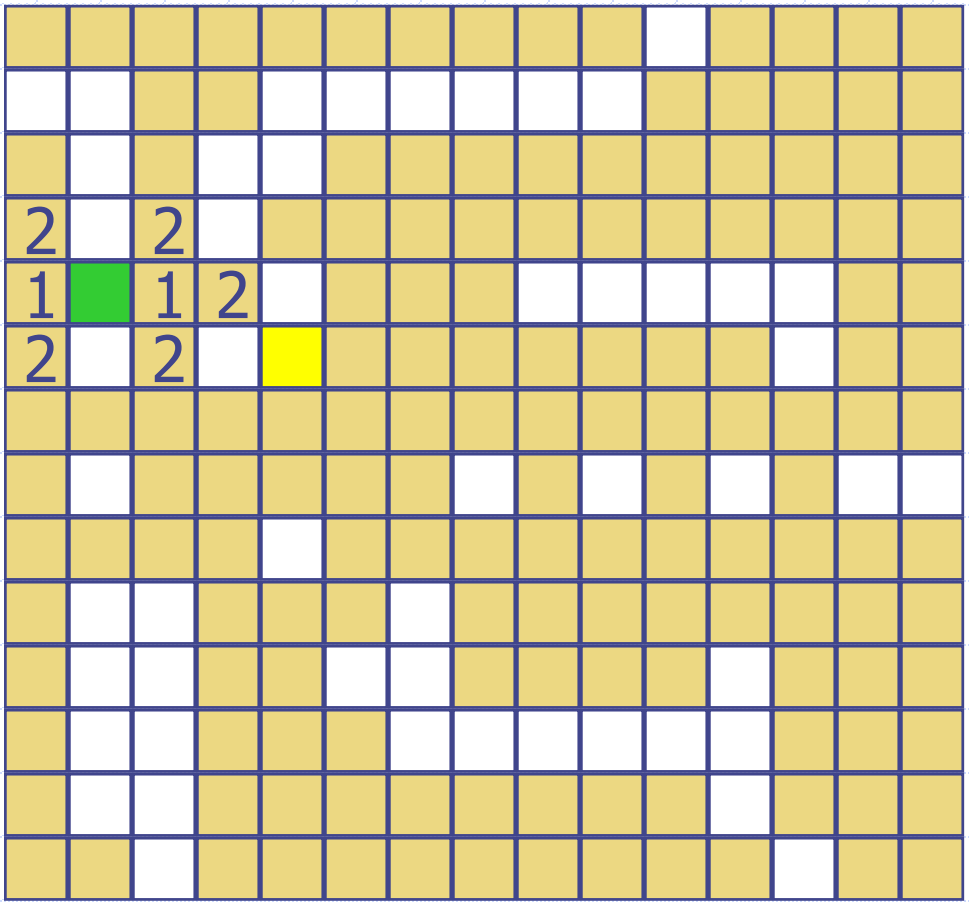


- همه مربع هایی برچسب نخورده که با فاصله دو واحد از start قابل دسترس هستند را برچسب بزنید

مسیریابی برای سیم

start pin

end pin

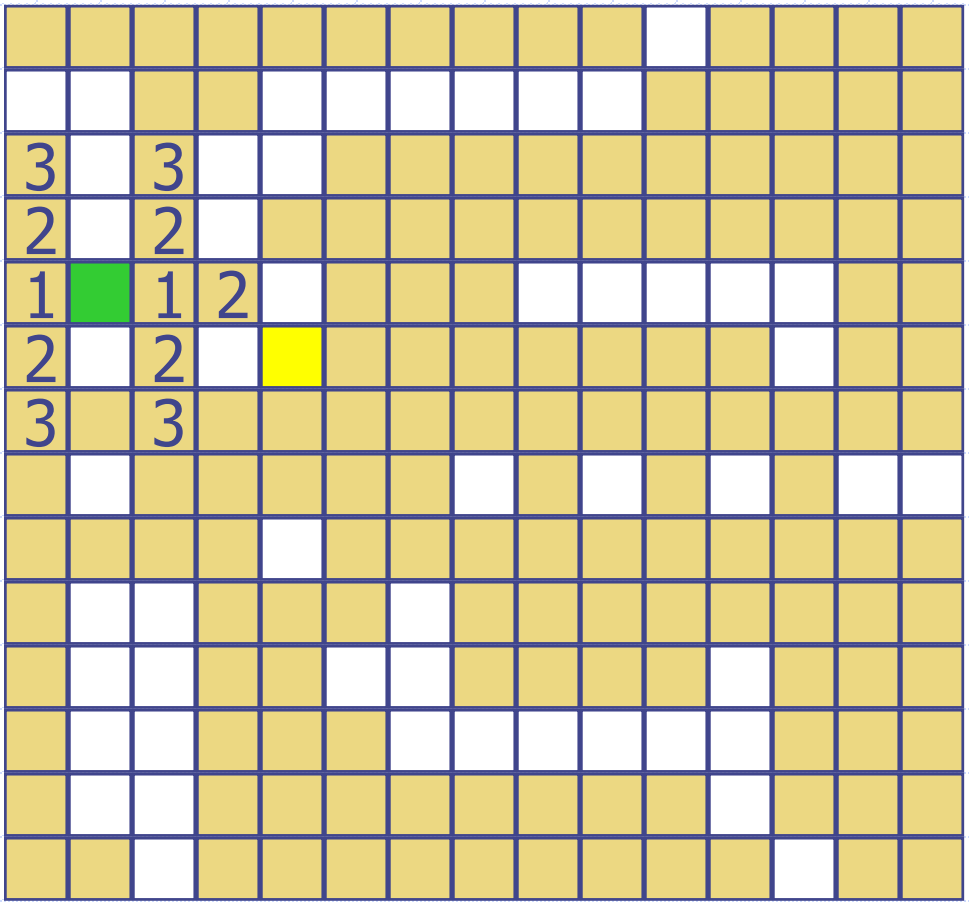


- همه مربع هایی برچسب نخورده که با فاصله سه واحد از start قابل دسترس هستند را برچسب بزنید

مسیریابی برای سیم

start pin

end pin

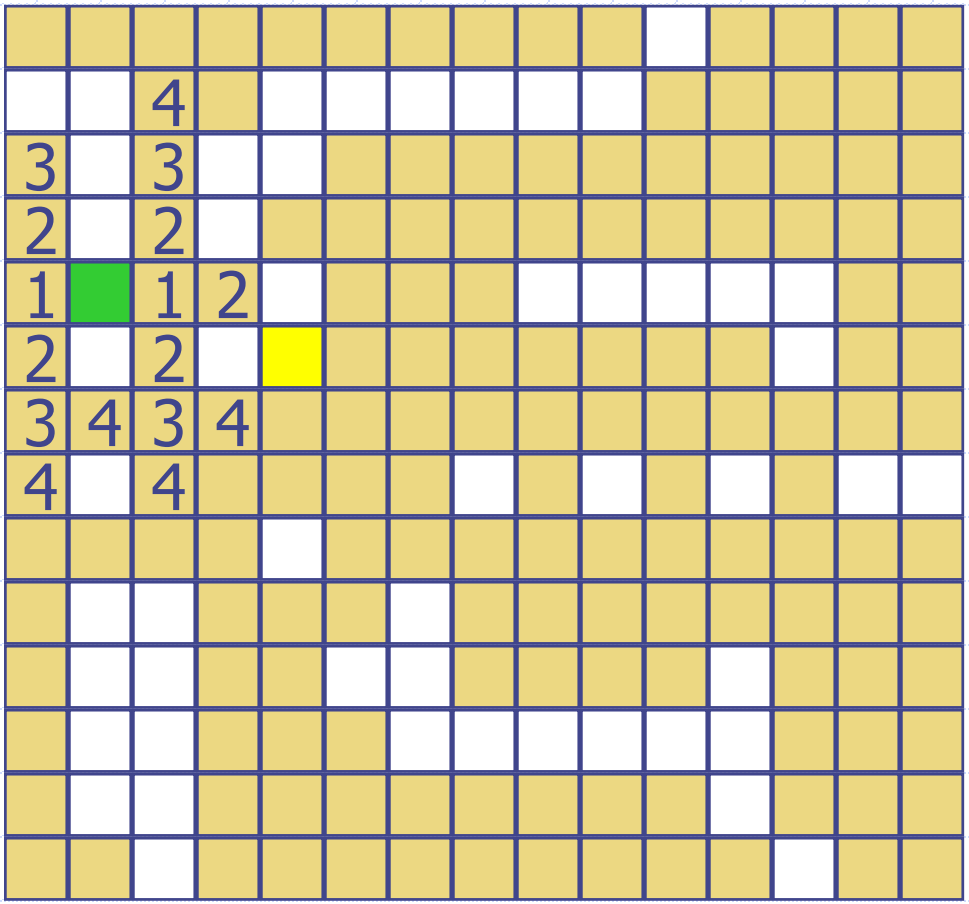


- همه مربع هایی برچسب نخورده که با فاصله چهار واحد از start قابل دسترس هستند را برچسب بزنید

مسیریابی برای سیم

start pin

end pin

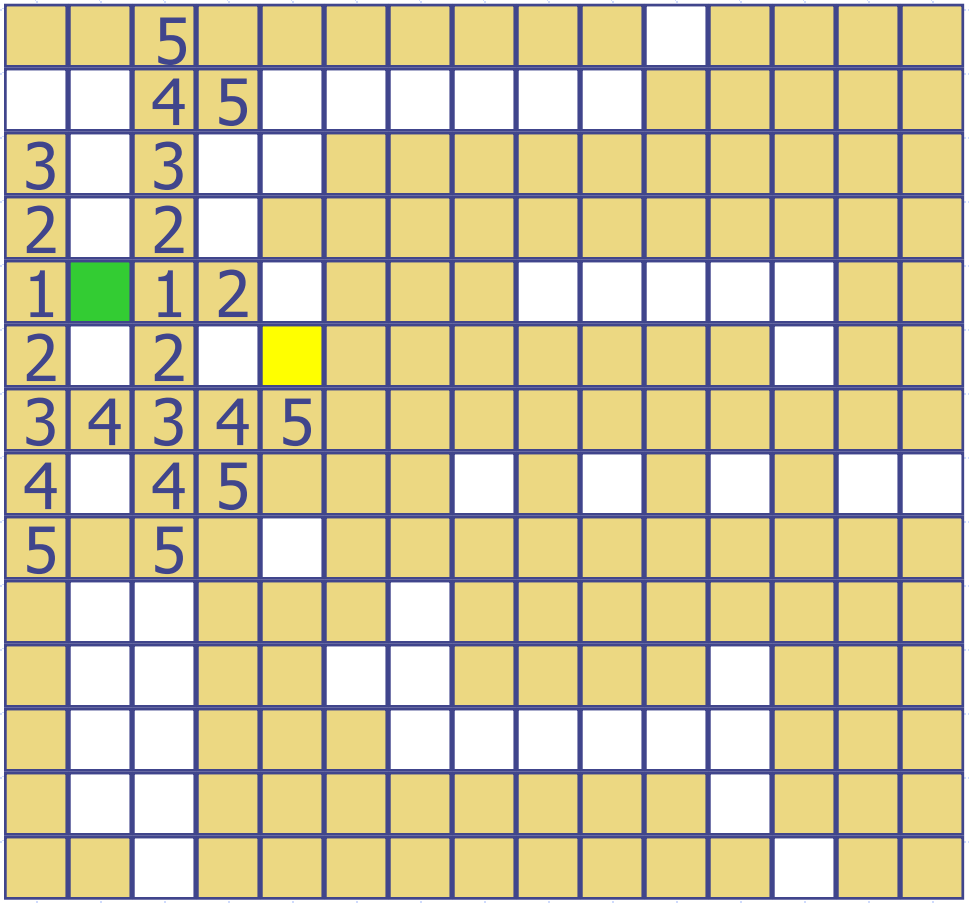


- همه مربع هایی برچسب نخورده که با فاصله پنج واحد از start قابل دسترس هستند را برچسب بزنید

مسیریابی برای سیم

start pin

end pin

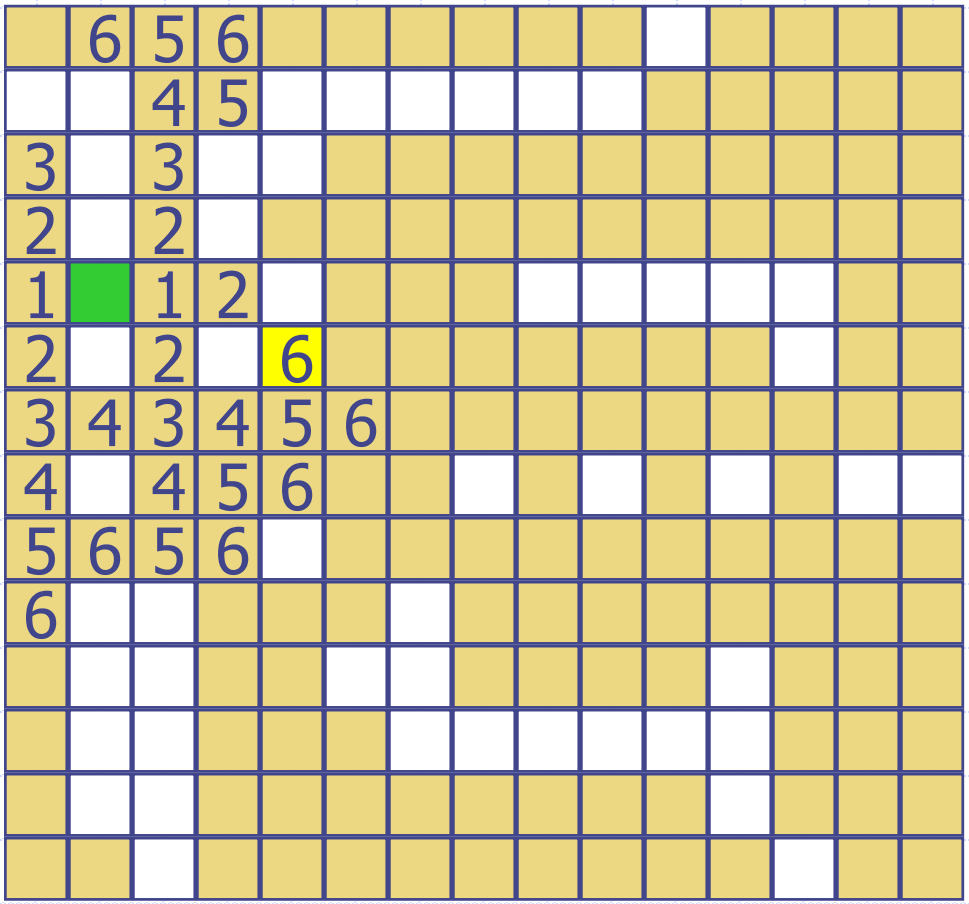


- همه مربع هایی برچسب نخورده که با فاصله شش واحد از start قابل دسترس هستند را برچسب بزنید

مسیریابی برای سیم

start pin

end pin

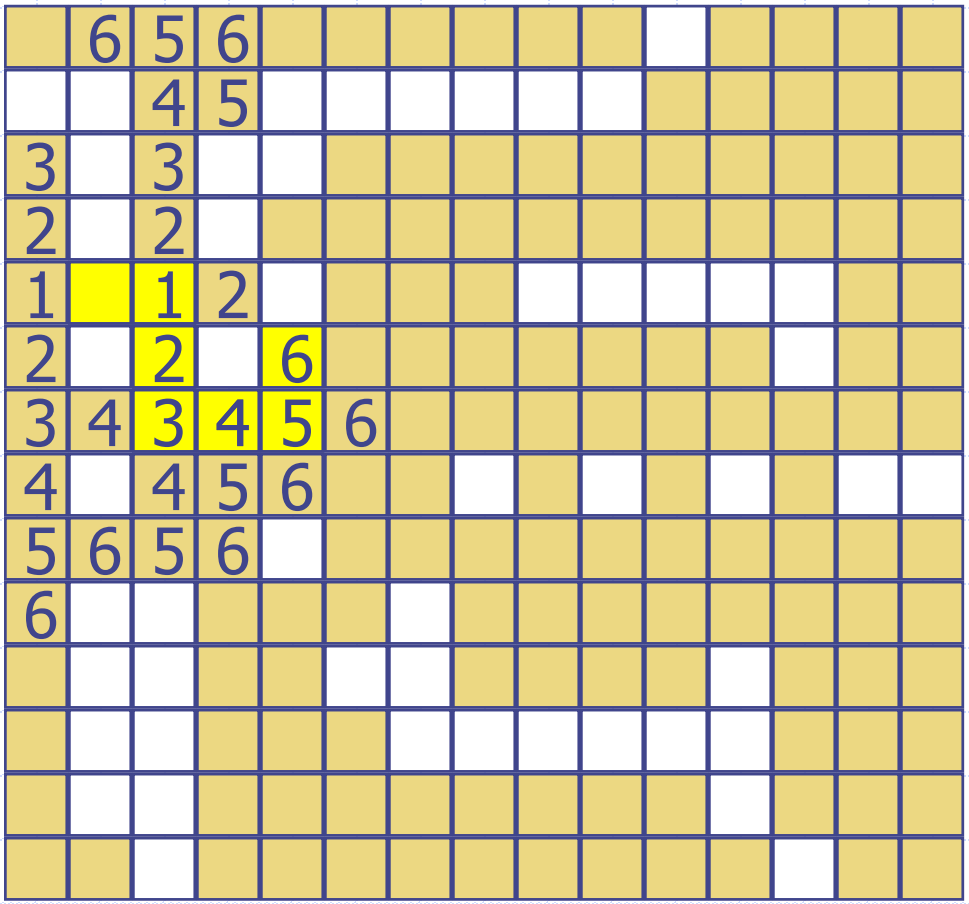


- به نقطه مورد نظر رسیده ایم مسیر را بازگردید

مسیریابی برای سیم

start pin

end pin



- به نقطه مورد نظر رسیده ایم مسیر را بازگردید

ارزیابی عبارات

- a = 4, b = c = 2, d = e = 3
- value of $x = a/b - c + d*e - a*c$
  - 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…

- 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؟

اولویت عملگرها در C

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

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

Postfix:
no parentheses,
no precedence

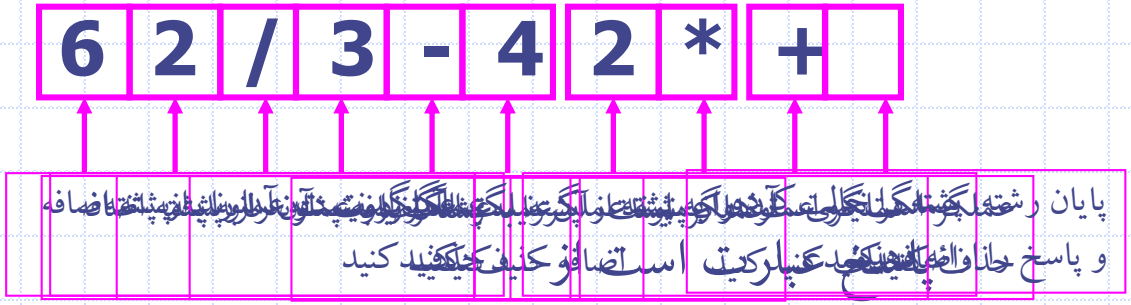
Infix	Postfix
2+3*4	2 3 4*+
*a***b* +5	*ab**5+
(1+2)*7	1 2+7*
*a***b* /*c*	*ab***c* /
((*a* /(*b* −*c* +*d*))*(*e* −*a*)**c*	*abc* −*d* +/*ea* −**c**
a /*b* −*c* +*d***e* −*a***c*	*ab* /*c* −*de**+*ac**−

ارزیابی عبارات

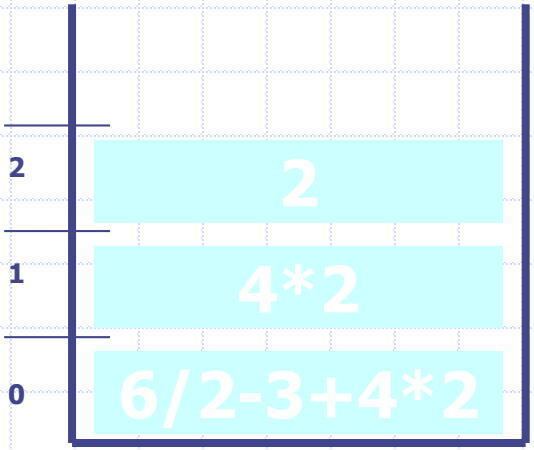
فقط یک پیمایش از چپ به راست از روی رشته انجام می شود

string: 6 2/3-4 2*+

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



6 / 2 - 3 + 4 * 2



اکنون باید top-1 شود

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

- Get Token

```
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 */
   }
}
```

ارزیابی عبارات

```
int eval(void)
{
/* evaluate a postfix expression, expr, maintained as a
global variable. '\0' is the 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);
         }
      }
      token = get_token(&symbol, &n);
   }
   return delete(&top); /* return result */
}
```

تبدیل عبارتهای میانوندی به پسوندی

رشته زیر را به فرم پسوندی تبدیل کنید

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

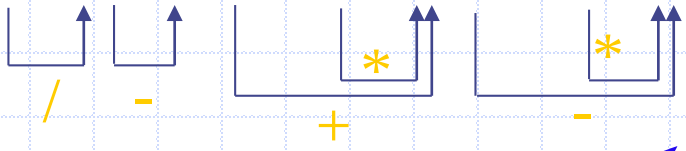
۱) عبارت را به صورت کامل پرانتزگذاری کنید

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

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	Stack [0]	[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	Stack [0]	[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**

تبدیل 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