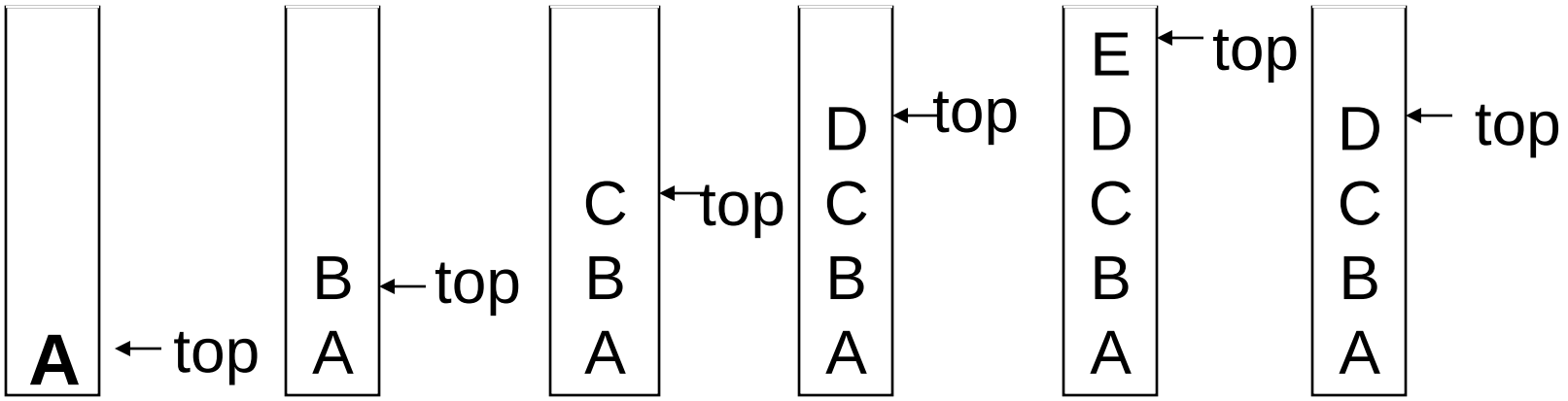
پشته و صف

پشته

❖ پشته یک لیست مرتب شده ای است که جایگذاری و حذف از یک سمت آن که top (بالا) نامیده می شود، صورت می گیرد.

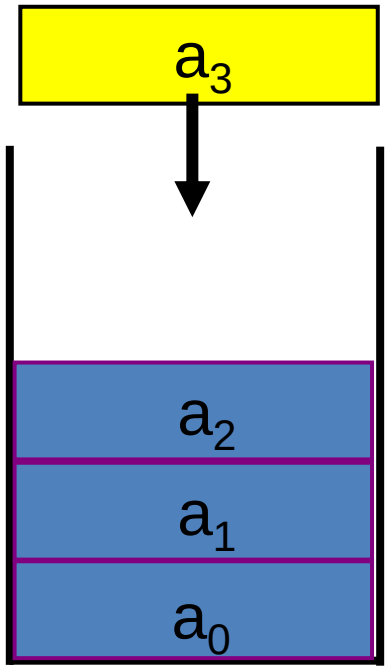
❖ محدودیت کار با پشته ما را ملزم می کند که اگر عناصر $A،B،C،D،E$ را به ترتیب به پشته اضافه کنیم ، E اولین عنصری خواهد بود که که از پشته حذف می گردد.

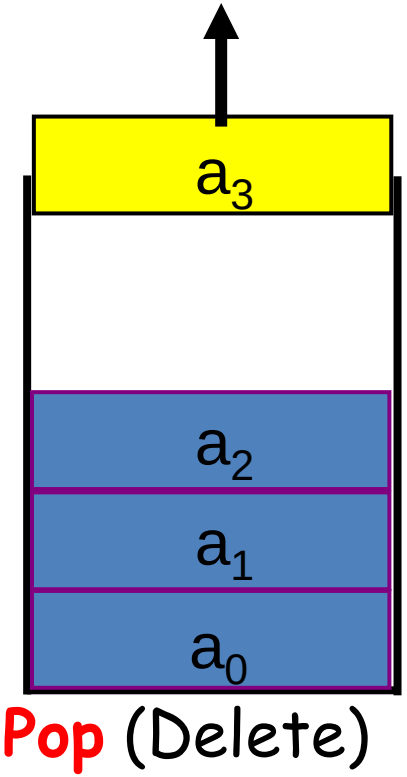


❖ از آنجا که آخرین عنصر وارده به پشته ، اولین عنصر حذف شده از آن می باشد ، پشته را به عنوان یک لیست $LIFO$ (آخرین ورودی ، اولین خروجی) می شناسیم.

پشته

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





ساختار نوع داده مجرد پشته

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**

ساختار نوع داده مجرد پشته

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.

پیاده سازی پشته

❖ راحت ترین روش پیاده سازی این ADT ، استفاده از یک آرایه یک بعدی به نام stack[MAX_STACK_SIZE] است که [MAX_STACK_SIZE] حداکثر تعداد عناصر آرایه می باشد.

❖ اولین یا پایین ترین عنصر پشته در stack[0] ذخیره ، دومی در stack[1] و i امین عنصر در stack[i-1] ذخیره می گردد.همراه با آرایه ، یک متغیر به نام top وجود دارد که به عنصر بالایی پشته اشاره می کند. در ابتدا به top مقدار ۱- داده می شود که نشان دهنده یک پشته خالی است. با این نمایش ، می توانیم عملکردهای ساختار را به صورت زیر پیاده سازی کنیم.

پیاده سازی پشته

Stack CreatS(max_stack_size)::=

```
# define MAX_STACK_SIZE 100 /* maximume 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;

اضافه کردن عنصر به پشته

```
Void  add (int *top ،element   item)
{
        if  (* top > = MAX_STACK_SIZE-1)   {
                stack_full();
                return;
         }
        stack [++*top] = item;
}
```

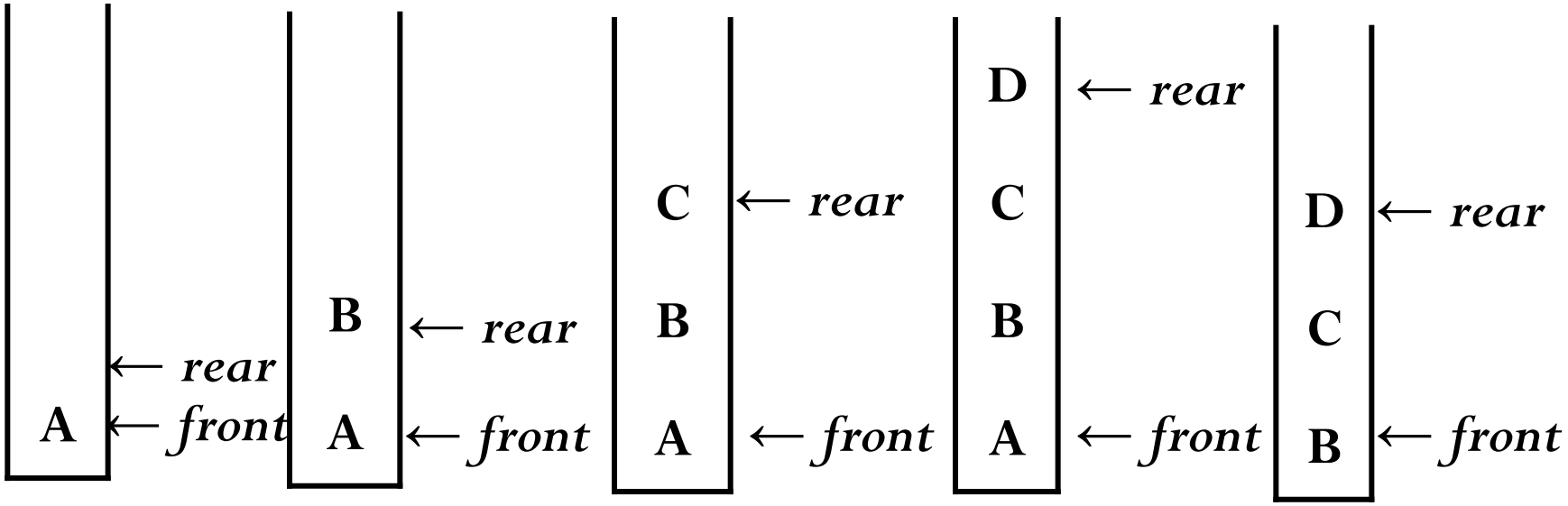
حذف کردن عنصر از پشته

```
element   delete (int  * top )
{
        if  (* top == -1)
              return  stack _empty ();
        else
              return stack [( * top)--];
}
```

صف

❖ صف یک لیست مرتب شده است که تمامی جایگذاری آن از یک سمت و تمام حذف های ان از سمت دیگر انجام می شود.

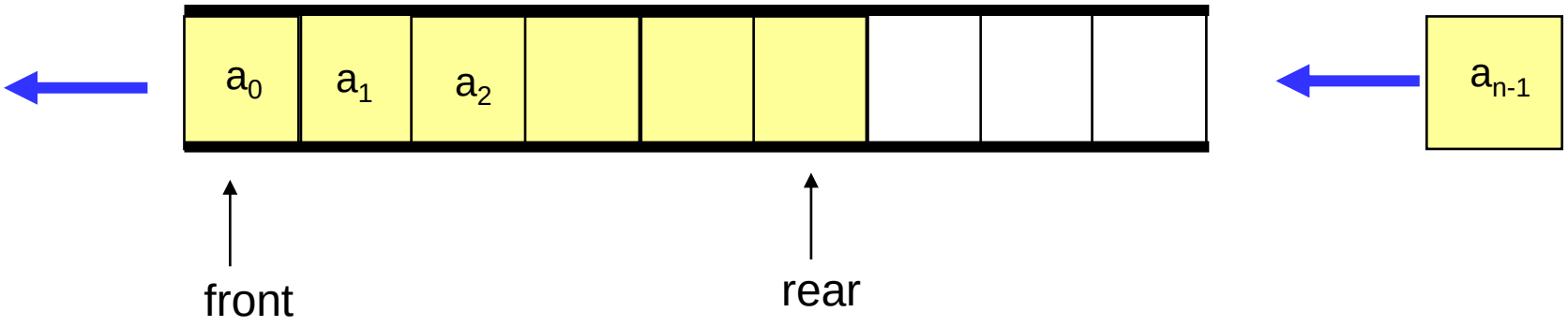
❖ محدودیت صف این است که ما A،B،C،D را به ترتیب اضافه می کنیم در حالی که A اولین عنصری است که حذف می شود.



❖ از آنجا که اولین عنصر وارده به صف، اولین عنصر حذف شده از آن می باشد ، صف را به عنوان یک لیست *FIFO*اولین ورودی ، اولین خروجی) می شناسیم.

صف

❖درصف $Q = a_0, a_1, ..., a_{n-1}$ ، a_0عنصر ابتدا (front) وa_{n-1} عنصر انتها (rear) می باشد وa_{i+1} در کنار a_i قرار دارد.



ساختار نوع داده مجرد صف

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*

ساختار نوع داده مجرد صف

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.

پیاده سازی صف

```
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 IsEmpty(queue) ::= front == rear
Boolean IsFullQ(queue) ::= rear == MAX_QUEUE_SIZE-1
```

اضافه کردن عنصر به صف

```
Void addq ( int * rear ،element  item )
{
/* add an  item  to the queue */
        if (* rear ==MAX_QUEUE _SIZE -1)  {
                queue_full();
                return;
        }
        queue[++* rear] = item;
}
```

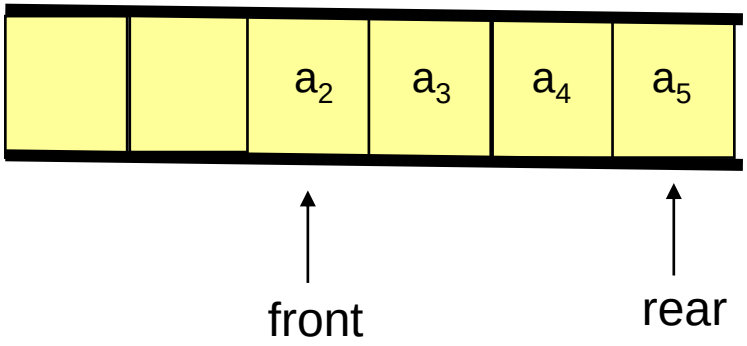
حذف عنصر از صف

```
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 حلقوی

❖ مشکلی که در صف داریم این است که پر بودن صف ممکن است اشتباهی تشخیص داده شود.

❖ ممکن است rear == MAX_QUEUE_SIZE-1 باشد ولی صف پر نباشد.



❖ برای مشکل صف معمولی از صف حلقوی استفاده می کنیم.

صف حلقوی

❖ پیاده سازی صف حلقوی همانند صف معمولی می باشد تنها تفاوتی که دارند این است که در هنگام تغییر مقدار front و rear از فرمول زیر استفاده می کنیم.

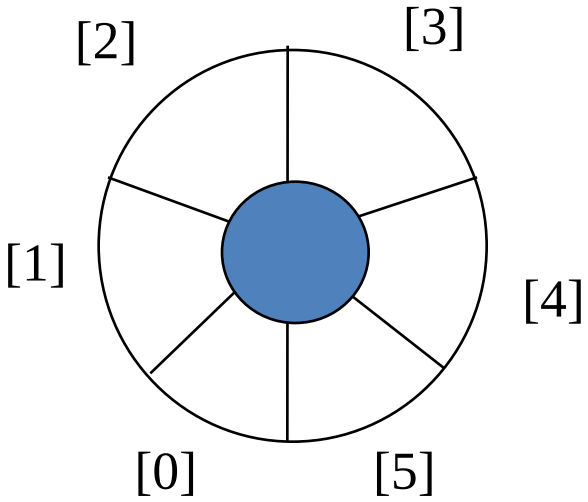
```
rear = (rear +1) % MAX_QUEUE_SIZE;

front = (front+1) % MAX_QUEUE_SIZE;
```

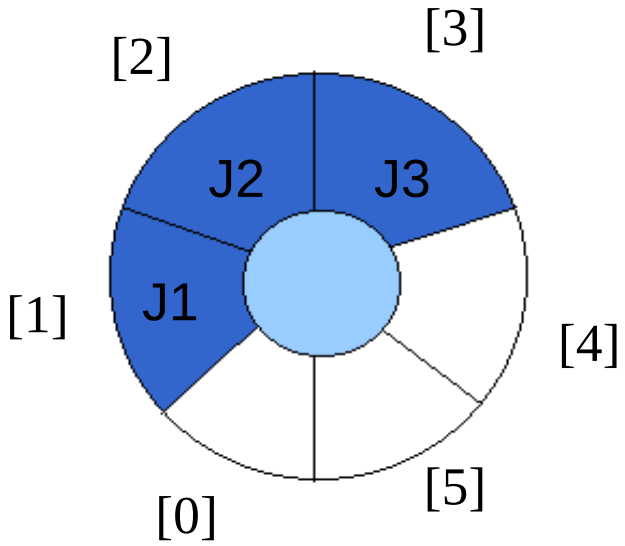
❖ به این صورت وقتی front یا rear به انتهای صف می رسد، دوباره به ابتدای صف بر می گردد.

❖ در صف حلقوی front همیشه به ابتدای صف و خانه خالی اشاره می کند ودر rear همیشه به انتهای صف و خانه پر اشاره می کند.

صف حلقوی

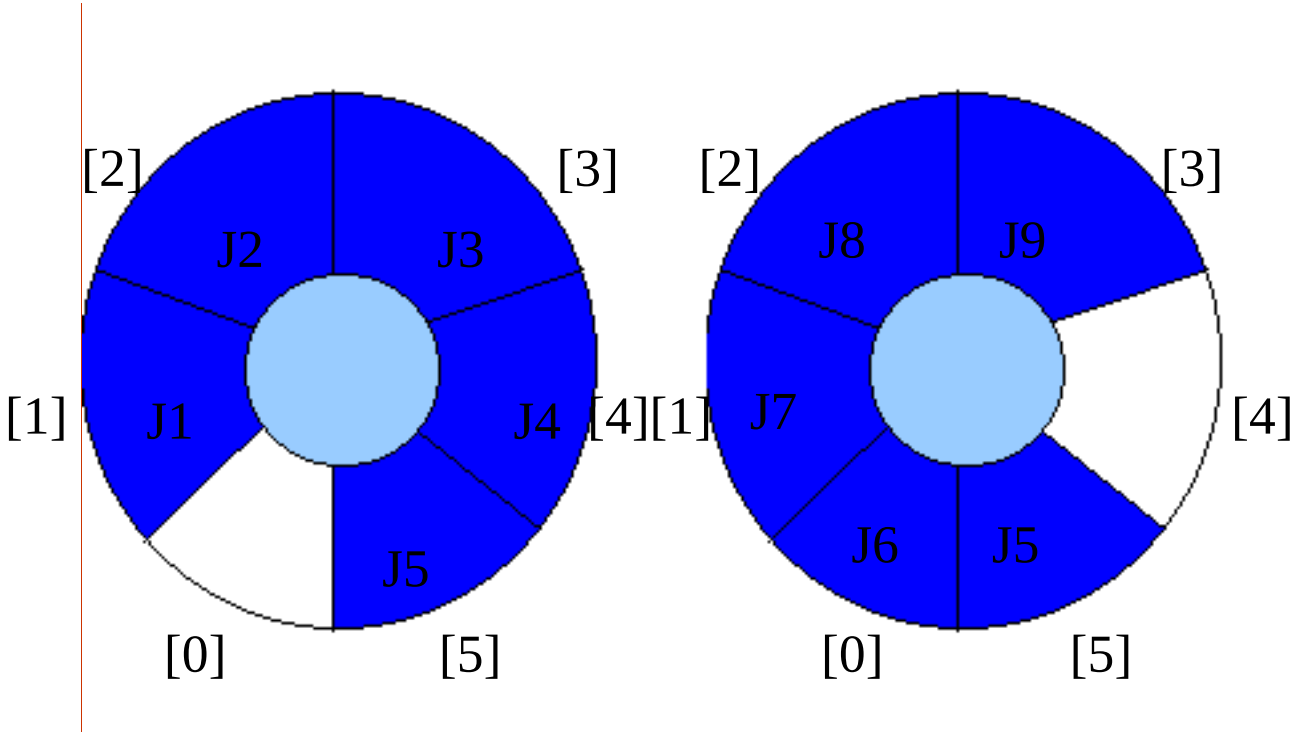


front = 0
rear = 0



front = 0
rear = 3

صف حلقوی



front =0
rear = 5

front =4
rear =3

اضافه کردن عنصر به 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;
}
```

حذف عنصر از صف حلقوی

```
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];
}
```

ارزشیابی عبارات

() []
! ~ ++ -- sizeof
- یکانی
* / %
+ -
<< >>
> >= < <=
= = !=
&
^
|
&&
||
?
= += -= *= %=
,

❖ در هر زبان برنامه سازی برای ارزیابی صحیح عبارات، به هر عملگر اولویتی نسبت می دهیم. حال در هر جفت پرانتز ، اول عملگرهایی که دارای اولویت بالاتر هستند ، ارزشیابی می شوند .

❖ شکل مقابل نشان دهنده اولویت عملگرها در زبان C می باشد.

روشهای نشانه گذاری عبارات

❖ روش infix(میانوندی): در این روش هر عملگر مابین دو عملوند مربوطه اش می آید.

(a+b)

❖ روش postfix (پسوندی): در این روش هر عملگر بعد از دو عملوند مربوطه اش می آید. در این روش پرانتز نداریم.

ab+

❖ روش prefix (پیشوندی): در این روش هر عملگر قبل از دو عملوند مربوطه اش می آید. در این روش پرانتز نداریم.

+ab

تبدیل عبارت infix به postfix (روش پرانتزگذاری)

❖ **به صورت زیر عمل می کنیم:**

۱- پرانتزگذاری کامل عبارات بر اساس تقدم عملگرها

۲- انتقال همه عملگرهای دودویی به نحوی که با پرانتز بسته مربوطه سمت راست آن تعویض شوند

۳- حذف تمام پرانتزها

2 + 3 * 4

تبدیل عبارت infix به prefix(روش پرانتزگذاری)

❖ **به صورت زیر عمل می کنیم:**

۱- پرانتزگذاری کامل عبارات بر اساس تقدم عملگرها

۲- انتقال همه عملگرهای دودویی به نحوی که با پرانتز باز مربوطه سمت چپ آن تعویض شوند

۳- حذف تمام پرانتزها

2 + 3 * 4

تبدیل عبارت infix به prefix و prefix با استفاده از روش پرانتزگذاری

❖ **مثال**: عبارات infix زیر را با استفاده از روش پرانتزگذاری به postfix و prefix تبدیل کنید.

infix	postfix	prefix
a * b + 5		
(1 + 2) * 7		
a * b / c		
((a / (b – c + d)) * (e – a) * c		
a / b – c + d * e – a * c		

ارزشیابی عبارت postfix با استفاده از پشته

❖ عبارات برای ارزیابی از چپ به راست پیمایش می شوند. عملوندها تا مشاهده یک عملگر، داخل پشته قرار می گیرند، سپس تعداد لازم از عملوندها را از پشته خارج و پس از انجام عملکرد مربوطه، نتیجه را دوباره به داخل پشته منتقل می کنیم . این کار را ادامه پیدا می کند تا به انتهای عبارت برسیم.

ارزشیابی عبارت postfix با استفاده از پشته

❖**مثال**: عبارت 82/3-42*+ را با استفاده از پشته ارزشیابی کنید.

Token	Stack [0]	[1]	[2]	Top
6	8			0
2	8	2		1
/	4			0
3	4	3		1
-	1			0
4	1	4		1
2	1	4	2	2
*	1	8		1
+	9			0
eos				

تبدیل عبارت infix به postfix با استفاده از پشته

تبدیل عبارت infix به postfix با استفاده از پشته

مثال: عبارت a+b*c را با استفاده از پشته به نشانه گذاری postfix تبدیل کنید.

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

تبدیل عبارت infix به postfix با استفاده از پشته

مثال: عبارت a*(b+c)*d را با استفاده از روش پشته به نشانه گذاری postfix تبدیل کنید.

Token	Stack [0]	Stack [1]	Stack [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*