آرایه‌ها Array

آرایـه نوعی ساختمان داده است که عناصر آن هم نوع بوده و هر یک از عناصر با یک اندیس به صورت مسـتقیم قابـل دسـتیابی اسـت. آرایـه مـی‌تواند یک بعدی , دو بعدی و یا چند بعدی باشد. آرایه‌های دو بعدی را با نام ماتریس می‌شناسیم.

$[L_1 \dots U_1 , L_2 \dots U_2 , L_n \dots U_n]$

Array [L … U] of items

تعداد عناصر آرایه = $U - L + 1$

تعداد عناصر آرایه n بعدی = $[U_1 - L_1 + 1][U_2 - L_2 + 1][U_n - L_n + 1]$

فضای اشغال شده توسط آرایه (فضای مورد نیاز) = $(U - L + 1) \times n$

مـثال : در یـک آرایـه بـه نام Float [200] اگر آدرس شروع آرایه در حافظه 1000 باشد A25 در کدام آدرس قرار دارد.

$A[i] = (i - L) \times n + \alpha$

محل عنصر iام در حافظه = $(25 - 0) \times 4 + 1000 = 1100$

آرایه‌های دوبعدی یا ماتریس‌ها به دو روش در حافظه ذخیره می‌شوند.

$$\begin{bmatrix} 2 & 5 \\ 1 & 6 \\ 3 & 4 \end{bmatrix} \quad 3 \times 2$$

۱. روش سطری Row Major

0	1	2	3	4	5
2	5	1	6	3	4

سطری

۲. روش ستونی Column Major

0	1	2	3	4	5
2	1	3	5	6	4

ستونی

A : Array $[L_1 \dots U_1 , L_2 \dots U_2]$ of items

تعداد عناصری = $[U_1 - L_1 + 1][U_1 - L_2 + 1]$

آدرس A[i , j] در روش سطری = $[(i - L_1) \times (U_2 - L_2 + 1) + (j - L_2)] \times n + \alpha$

آدرس A[i , j] در روش ستونی = $[(j - L_2) \times (U_1 - L_1 + 1) + (i - L_1)] \times n + \alpha$

مثال : طبق آرایه زیر , آدرس‌های خواسته شده را محاسبه نمائید.

$L_1 \dots U_1 \ L_2 \dots U_2$

A : [1 … 3 , 1 … 2] ==➔ در زبان C داریم ==➔ A[3][2]

$$\begin{bmatrix} 2 & 5 \\ 1 & 6 \\ 3 & 4 \end{bmatrix}$$

روش سطری $A[3,2] = (3-1) \times (2-1+1) + (2-1) = 2 \times 2 + 1 = 5$

روش ستونی $A[3,2] = (2-1) \times (3-1+1) + (3-1) = 1 \times 3 + 2 = 5$

روش سطری $A[1,2] = (1-1) \times (2-1+1) + (2-1) = 1$

روش ستونی $A[1,2] = (2-1) \times (3-1+1) + (1-1) = 3$

تمرین : در یک آرایه به شکل A [1 ... 100 , 1 ... 26] of integer اگر این آرایه از محل 1000 حافظـه شروع شده باشد محل داده A [60 , 6] در روش سطری و محل داده A [20 , 4] در روش ستونی کدام آدرس حافظه است.

$A[60,6] = (60-1) \times (26-1+1) + (6-1) \times 2 + 1000 = 4078$

$A[20,4] = (4-1) \times (100-1+1) + (20-1) \times 2 + 1000 = 1638$

در آرایـه‌هـای دو بعـدی مربعی یا ماتریس‌های مربعی که کلیه عناصر بالای قطر اصلی آن صفر باشند یـک ماتریس پایین مثلثی تشکیل می‌گردد و برعکس اگر کلیه عناصر پایین قطر اصلی آن صفر باشند یـک ماتـریس بـالا مثلثـی تشـکیل خـواهد شـد. در یـک ماتـریس پایین مثلثی یا بالا مثلثی حداکثر $\frac{n(n+1)}{2}$ عنصر غیر صفر داریم که n اندازه هر بعد ماتریس است.

$$\begin{bmatrix} 1 & 6 & 7 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{bmatrix}$$ بالا مثلثی $= \frac{3(3+1)}{2} = 6$ حداکثر عناصر غیر صفر

$A[i,j] = 0 \quad i > j \Longrightarrow$ ماتریس بالا مثلثی

$A[i,j] = 0 \quad i < j \Longrightarrow$ ماتریس پایین مثلثی

اگـر انـدازه ابعـاد ماتریس‌های مثلثی افزایش یابند این ماتریس‌ها حاوی تعداد زیادی صفر خواهند بود کـه ذخیـره کردن سطری یا ستونی ماتریس به طور کامل در حافظه باعث هدر رفتن بخشی از فضای حافظـه مـی‌گردد. به همین دلیل ماتریس‌های مثلثی را بصورت سطری یا ستونی بدون در نظر گرفتن صفرها در حافظه ذخیره می‌کنند.

پایین مثلثی =====> سطری $\frac{(i-1)\times i}{2}+j$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

=====>

1	2	3	4	5	6
1	2	5	3	1	1

پایین مثلثی

بالا مثلثی =====> ستونی $\frac{(j-1)\times j}{2}+i$

$$\begin{bmatrix} 1 & 6 & 7 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{bmatrix}$$

=====>

1	2	3	4	5	6
1	6	2	7	5	4

بالا مثلثی

جمع ماتریس‌ها

در جمـع دو ماتریس , حتماً باید یک ماتریس $m \times n$ با یک ماتریس $m \times n$ جمع شده و نتیجه نیز یک ماتریس $m \times n$ خواهد شد. در این عملیات عناصر دو آرایه نظیر به نظیر با یکدیگر جمع خواهند شد.

$$A_{m\times n} + B_{m\times n} = C_{m\times n}$$

$$\text{for } (i = 0 \,,\, i < m \,,\, ++i)$$

$$\text{for } (j = 0 \,,\, j < n \,,\, ++j)$$

$$C_{ij} = a_{ij} + b_{ij}$$

ضرب ماتریس‌ها

در عمـل ضـرب , یـک ماتـریس A_{mL} و یک ماتریس B_{Ln} با یکدیگر ضرب شده و ماتریس بدست آمده نیز دارای سطر و ستونهایی می‌باشد که سطر ماتریس بدست آمده با تعداد سطرهای ماتریس اول و ستون ماتریس بدست آمده با تعداد ستونهای ماتریس دوم برابر است.

$$C_{mn} = A_{\underset{i}{m}\,\underset{k}{L}} \times B_{\underset{k}{L}\,\underset{i}{n}}$$

$$\begin{bmatrix} 3 & 4 & 1 & 2 \\ 2 & 5 & 3 & 1 \\ 1 & 0 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} - & - \\ - & - \\ - & - \end{bmatrix}$$

$$3 \times 4 \quad \times \quad 4 \times 2 \quad = \quad 3 \times 2$$

$$\text{for } (i = 0 \,,\, i < m \,,\, ++i)$$

$$\text{for } (j = 0 \,,\, j < n \,,\, ++j)$$

$$\{$$

$$C_{ij} = 0$$

$$\text{for } (k = 0 \,,\, k < L \,,\, ++k)$$

$$C_{ij} = a_{ik} \times b_{kj} + C_{ij}$$

$$\}$$

تمرین : مقدار C [1 ,0] را در حاصلضرب دو ماتریس مثال قبل بدست آورید.

جــواب : برای بدست آوردن مقدار خواسته شده باید حلقه‌های for بالا را Trace کنیم. پس بنابراین داریم :

$C_{ij} = 0$

$C_{ij} = a_{ik} \times b_{kj} + C_{ij}$

$C_{ij} = 2 \times 1 + 0 = 2$

$C_{ij} = 5 \times 0 + 2 = 2$

$C_{ij} = 3 \times 1 + 2 = 5$

$C_{ij} = 1 \times 3 + 5 = 8$

i	j	k	L	C_{ij}
1	0	0	4	0
		1		2
		2		2
		3		5
				8

ترانهاده

برای اینکه ترانهاده یک ماتریس را بدست آوریم جای سطرها و ستونهای ماتریس عوض می‌شوند.

$$A\begin{bmatrix} 2 & 3 & 5 \\ 1 & 0 & 7 \end{bmatrix} \rightarrow A^T \begin{bmatrix} 2 & 1 \\ 3 & 0 \\ 5 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & 4 \\ 1 & 2 & 5 \\ 1 & 3 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & 2 & 3 \\ 4 & 5 & 8 \end{bmatrix}$$

$A_{m \times n}$

for (i = 0 , i < m , + + i)

for (j = 0 , j < n , + + j)

A [j] [i] = A [i] [j]

جستجوی خطی در آرایه

```
Array A[n] , x
Int search (A[n] , x) ;
{
    int i = 1 ;
    while (i <= n && A[i] != x)
        i + + ;
    if (i > n ) return – 1 // داده پیدا نشده
    else return i // داده در محل اندیس آرایه است
}
```

1	2	3	4	5
1	5	2	8	6

n	x	i	A[i]
5	8	1	1
		2	5
		3	2
		4	8

x = A[4]

جستجوی دودویی برای آرایه‌های مرتب

```
Int bsearch (A[n] , int x , int L , int U)
{
    int i ;
    while
    {
        i = ⌈(L + U) / 2⌉;
        if ( x < A[i] ) U = i – 1
        else if ( x > A[i] ) L = i + 1
            else return i // داده در اندیس i است
    }
    return – 1 // داده پیدا نشده است
}
```

x	i	L	U	A[i]
8	5	1	10	12
	2	3	4	2
	4	4		5
	3			8

جمع دو چندجمله‌ای بوسیله آرایه

$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$

a_n	a_{n-1}	a_{n-2}	....	a_1	a

$P(x) = 3x^2 + 5x + 2$

0	3	5	2

+

$P(x) = 3x^3 + 3$

3	0	0	3

=

3	3	5	5

$P(x)\ x^{100} + 2$

ضریب	1	2
توان	100	0

Stack یا پشته

Stack لیستی است که اعمال ورودی و خروجی یا اضافه و حذف در آن از یک طرف لیست انجام می‌شود. به این جهت به آن لیست Last In First Owt (LIFO) می‌گویند. بدین معنی که آخرین ورودی به پشته , اولین خروجی خواهد بود. عنصر بالایی پشته را top پشته می‌گویند. با افزودن داده روی پشته , متغیر top یکی زیاد شده و داده در محل top از پشته قرار می‌گیرد. برای خارج کردن یک عنصر از پشته نیز داده‌ای که در محل top قرار گرفته از Stack خارج می‌گردد و متغیر top یکی کم می‌شود. مقدار اولیه top صفر است و با افزودن داده به یک پشته n عضوی , top می‌تواند تا مقدار n تغییر کند.

Top = 0 ===➔ پشته خالی است

Top = n ===➔ پشته پر است

دو عمل اصلی برای پشته‌ها را با push کردن و pop کردن می‌شناسیم. Push (x) داده x را در بالای پشته قرار می‌دهد و عمل pop عنصر بالای پشته را در متغیر x ذخیره می‌کند.

x = pop ≡ pop (x)

Stack : Array [1 .. n] of items

```
int pop ( )
{
    int x ;
    if (top = 0)
    {
        C out << " پشته خالی است " ;
        return – 1 ;
    }
    else
    {
        x = Stack [top] ;
        top = top – 1 ;
    }
    return x ;
}
```

```
void push (int x)
{
    if (top = = n)
    {
        C out << " پشته پر است " ;
        return – 1 ;
    }
    else
    {
        top + + ;
        Stack [top] ;
    }
}
```

مثال : مقدار نهایی A و B و C چقدر است؟

n = 5 A = 10 B = 2 C = 5

push (B)

push (A + B)

pop (C)

push (A – B)

push (C)

push (B)

pop (A)

pop (B)

push (A × B)

push (C)

push (A)

pop (B)

pop (C)

pop (A)

2	
2	12
12	24
12	8
2	

A	B	C
~~10~~	~~2~~	~~5~~
~~2~~	~~12~~	~~12~~
24	2	12

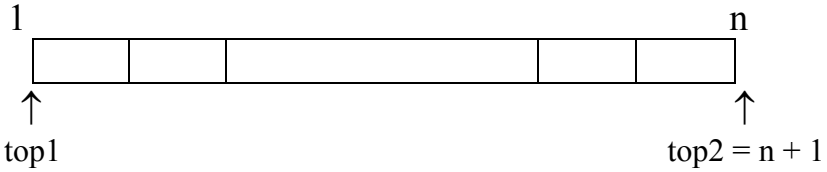
پشته‌های چندگانه

پشته دوگانه : برای پیداسازی دو پشته در یک آرایه نیاز به دو متغیر top1 برای نشان دادن بالاترین عنصر پشته اول و top2 برای بالاترین عنصر پشته دوم داریم. top1 و top2 در جهت عکس یکدیگر حرکت می‌کنند. مقدار اولیه top1 = 0 و مقدار اولیه top2 = n + 1 است.

top1 = 0 ===➔ پشته ۱ خالی است

top2 = n + 1 ===➔ پشته ۲ خالی است

top2 = top1 + 1 ===➔ آرایه پر است



دنباله‌های قابل قبول در پشته‌ها

هرگاه اعدادی را به صورت مرتب شده صعودی داشته باشیم و بخواهیم اعداد دیگری را از آن استخراج کنیم باید این قانون را راعایت کنیم که اعداد بزرگتر در صورتیکه اعداد کوچکتر در پشته قرار نگرفته‌اند حق قرار گرفتن در پشته را ندارند. مثلاً اعداد ۴ , ۳ , ۲ , ۱ را در نظر می‌گیریم. عدد ۲ در صورتی می‌تواند push شود که حتماً عدد ۱ push شده باشد و عدد ۳ زمانی می‌تواند push شود که اعدا ۱ و ۲ قبلاً push شده باشند.

مثال : چهار عدد ۴ , ۳ , ۲ , ۱ را داریم. کدامیک از اعداد زیر را می‌توانیم تولید کنیم ؟

۴ ۳ ۱ ۲	۴ ۲ ۳ ۱	۳ ۲ ۴ ۱	۳ ۱ ۴ ۲	۲ ۱ ۳ ۴
push 1 push 2 push 3 push 4 pop 4 pop 3	push 1 push 2 push 3 push 4 pop 4	push 1 push 2 push 3 pop 3 pop 2 push 4 pop 4 pop 1	push 1 push 2 push 3 pop 3	push 1 push 2 pop 2 pop 1 push 3 pop 3 push 4 pop 4
قابل تولید نیست	قابل تولید نیست		قابل تولید نیست	

اگر اعداد به صورت صعودی داده شوند (۴ ۳ ۲ ۱) و سه عدد a و b و c داشته باشیم بطوریکه b < c < a در اینصورت دنباله abc قابل تولید نیست.

ارزشیابی عبارت

اولویت عملگر

بطور کلی اگر عبارت $a \times b + c / d$ را داشته باشیم اولویت عملگرها را به صورت زیر می‌نویسیم :

1. ( )
2. Not , – (قرینه) , توان
3. and , × , / , mod
4. OR , + , –
5. < , > , <= , >= , <> (! =)

نکته : بین عملگرهایی که اولویت مساوی دارند عملگری زودتر محاسبه می‌گردد که سمت چپ باشد.

روش نمایش عبارات محاسباتی

میانوندی	infix	a + b
پسوندی	postfix	ab +
پیشوندی	prefix	+ ab

تبدیل عبارات میانوندی به پسوندی و پیشوندی بدون استفاده از پشته

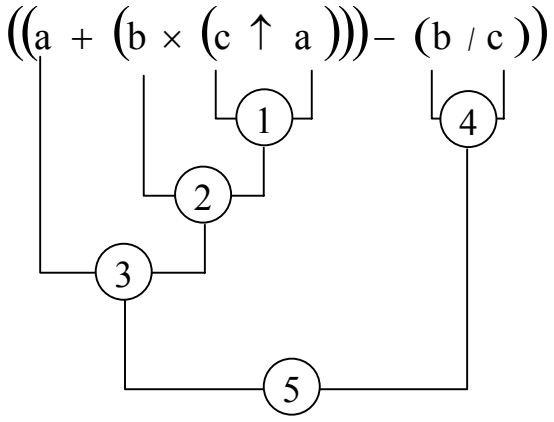
۱- پرانتز گذاری

۲- برای تبدیل به پیشوندی , درون هر پرانتز عملگر را به سمت چپ منتقل می‌کنیم.

۳- برای تبدیل به پسوندی , درون هر پرانتز عمگلر را به سمت راست منتقل می‌کنیم.

۴- پرانتزها را حذف می‌کنیم.

مثال :



$$\text{postfix} = \big(a(b(ca))\uparrow\times+(bc)/\big) = abca\uparrow\times+bc/-$$

$$\text{prefix} = -+a\times b\uparrow ca/bc$$

استفاده از پشته در تبدیل عبارات infix به postfix

۱- عبارت infix را از چپ به راست پیمایش می‌کنیم.

۲- پرانتز باز را در پشته push می‌کنیم.

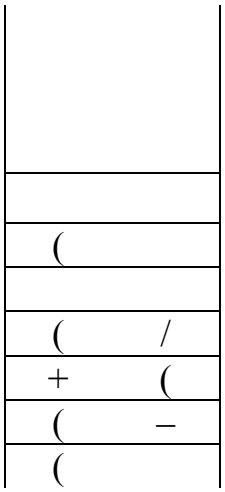
۳- عملوندها را در خروجی می‌نویسیم.

۴- در صورتیکه به یک عملگر رسیدیم اگر top پشته دارای عملگری با اولویت بیشتر یا مساوی نبود آنرا push می‌کنیم در غیر اینصورت عملگر top پشته را pop کرده و در خروجی می‌نویسیم.

۵- هرگاه به پرانتز بسته رسیدیم آنقدر pop می‌کنیم تا به اولین پرانتز باز برسیم.

مثال :

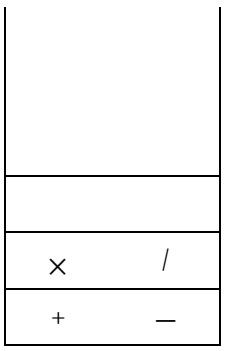
$$((a + (b \times (c \uparrow a))) - (b / c))$$



$abca \uparrow \times + bc / -$

مثال : با استفاده از پشته , عبارت زیر را به صورت postfix بنویسید.

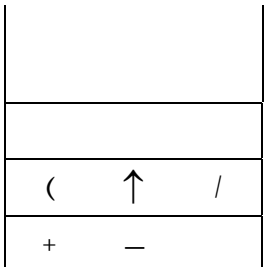
$$a + b \times c \uparrow a - b / c$$



$abca \uparrow \times + bc / -$

مثال : با استفاده از پشته , عبارت زیر را به صورت postfix بنویسید.

$$a+(b\times c)\uparrow a-b/c$$



$$abc\times a\uparrow +bc/-$$

برای تبدیل عبارات infix به عبارات prefix از دو پشته استفاده می‌کنیم. یکی پشته عملوندها و دیگری پشته عملگرها. push کردن و pop کردن در پشته عملگرها مانند تبدیل infix به postfix است. با رسیدن به هر عملوند آنرا در پشته عملوندها push می‌کنیم. در صورت pop شدن هر عملگر از پشته عملگرها , دو عملوند بالای پشته عملوندها pop شده و با عملگر مربوطه به شکل prefix در پشته عملوندها push می‌شود. بقیه قوانین مانند قوانین infix به postfix است.

مثال : عبارت زیر را بوسیله پشتبه از infix به prefix بنویسید.

$$a+(b\times c)\uparrow d/a-c\times b$$

×			
(	↑	/	×
+	-		

c	d	a	d
b	×bc	↑×bcd	/↑×bcda
a	+a/↑×bcda		

$$-+a/\uparrow\times bcda\times cd$$

مثال : عبارت infix زیر را بوسیله دو پشته به prefix تبدیل کنید.

$$a+b\times c\uparrow(2-b)\times c/(d+a)$$

-		
(	+	
↑	(	
×	×	/
+		

b				
2	-2b	a		
c	↑c-2b	c	d	+da
b	×b↑c-2b	××b↑c-2b	/××b↑c-2bc+da	
a	+a/××b↑c-2bc+da			

تبدیل عبارت postfix به infix

با استفاده از یک Stack می‌توان رشته postfix ورودی را به infix تبدیل کرد. برای این منظور

رشته postfix را از چپ پردازش می‌کنیم. هر عملوند درون پشته push می‌شود. با رسیدن به هر عملگر , دو عنصر پشته pop شده و بصورت infix نوشته می‌شود. سپس عبارت infix تولید شده درون پشته push می‌شود. در پایان پردازش رشته ورودی , پشته حاوی یک عنصر است که شکل infix مورد نظر می‌باشد. خروجی infix باید لزوماً پرانتزگذاری شده باشد. عملوند top پشته سمت راست عملگر نوشته می‌شود.

مثال :

abca↑ ×+bc/–

a			
c	c↑a	c	
b	b×(c↑a)	b	b/c
a	a+(b×(c↑a))	(a+b×(c↑a))–(b/c))	

تبدیل عبارات prefix به infix

برای تبدیل عبارت prefix به infix باید رشته ورودی را از سمت راست پردازش کنیم. مانند روش قبل عملوندها در پشته push می‌شوند و با رسیدن به هر عملگر , دو عملوند بالای پشته pop شده و با عملگر ورودی بصورت infix نوشته می‌شود و نتیجه در پشته push می‌شود. عملوند top پشته سمت چپ عملگر قرار می‌گیرد.

–+a×b↑ca/bc

c	b	a		
b	a	(c↑a)	(b×(c↑a))	(a+(b×(c↑a))
c	(b/c)	(a+b×(c↑a) –(b/c)		

تبدیل عبارات postfix به prefix و بالعکس

برای تبدیل عبارات postfix و prefix به همدیگر می‌توان آنها را ابتدا تبدیل به حالت میانی infix کرده و سپس عبارت infix را با روشهای گفته شده به حالت مطلوب تبدیل نمود. همچنین می‌توان بصورت مستقیم عبارت postfix و prefix را با استفاده از الگوریتم قبلی به یکدیگر تبدیل کرد. با این تفاوت که هنگامیکه در حین پردازش رشته ورودی به یک عملگر رسیدیم , دو عملوند بالای پشته pop شده و به جای اینکه به infix با عملگر ورودی در پشته push شوند به هر کدام از حالتهای مورد نظر postfix یا prefix در پشته push می‌شوند.

صف (queue)

صف لیستی است که عمل افزودن داده‌ها درون آن از یک طرف لیست یا انتهای لیست و عمل حذف داده‌ها از سمت دیگر یا ابتدای لیست انجام می‌شود. صف را لیست FIFO (First In First Out) می‌نامند. زیرا اولین عنصر ورودی , اولین عنصر خروجی از صف نیز هست. در ساختمان داده صف دو متغیر front و rear به ترتیب برای نشان دادن جلو و انتهای صف بکار می‌روند. صف را می‌توان با استفاده از آرایه‌ها یا لیست‌های پیوندی پیاده سازی کرد.

اگر صف را آرایه‌ای n عضوی از عناصر بدانیم مقادیر front و rear می‌تواند از صفر تا n تغییر کند که برای صف در ابتدا مقادیر اولیه صفر را برای front و rear تعریف می‌کنیم. front = rear = 0

در صورتیکه متغیر front با rear برابر باشد صف خالی است و در صورتیکه rear برابر با n باشد صف پر است.

rear = n ===➔ صف پر است

front = rear ===➔ صف خالی است

دو عمل اصلی برای صف , حذف کردن داده‌ها از صف و افزودن داده‌ها به صف است که به ترتیب با delqueue و Addqueue نمایش می‌دهیم. تابع Addqueue(x) به این معنی است که عنصر x به انتهای صف اضافه شده است و delqueue نیز مقدار جلوی صف را برداشته و در متغیر x قرار می‌دهد. x = delqueue

پیاده سازی تابع Addqueue و delqueue از صف

queue : Array [1 .. n] of item

برای اضافه کردن

```
void Addqueue (int x)
{
    if (rear = = n)
        C out << " صف پر است " ;
    else
    {
        rear + + ;
        queue[rear] = x ;
    }
}
```

برای حذف کردن

```
int delqueue ( )
{
    if (front = = rear)
    {
        C out << " صف خالی است " ;
        return 0 ;
    }
    else
    {
        front + + ;
        x = queue[front] ;
        return x ;
    }
}
```

مثال : با استفاده از توابع صفحه قبل مقادیر نهایی A و B و C را بدست آورید.

A = 5 B = 10 C = 2 n = 4

Addqueue (A + B)

Addqueue (C)

Addqueue (B × C)

A = delqueue ( )

B = delqueue ( )

Addqueue (B – A)

15	2	20	–13
1	2	3	4

Rear	Front	A	B	C
0	0	5	10	**2**
1	1	**15**	**2**	
2	2			
3				
4				

پس بنابراین داریم : **A = 15** **B = 2** **C = 2**

توابع Addqueue و delqueue به صورتیکه نوشته شد یک صف خطی را پیاده‌سازی می‌کنند. مشکل صف خطی این است که تنها یک بار قابل پر شدن است و در صورتیکه عناصر آن حذف شوند نیز با پیغام « صف پر است » مواجه می‌شوید به همین دلیل صف را بصورت حلقوی تعریف می‌کنیم. در صف حلقوی (دوار) rear و front بعد از رسیدن به آخرین مقدار خود در صورت وجود شرایط لازم مجدداً مقادیر اولیه را می‌توانند بگیرند. صف حلقوی n عضوی را بصورت آرایه صفر تا n – 1 تعریف می‌کنیم.

queue : Array [0 .. n – 1] of item

در این حالت وقتی rear = n – 1 عنصر بعدی در queue[0] قرار می‌گیرد. در صف حلقوی front = rear به معنای خالی بودن صف است ولی شرط پر بودن صف بدین ترتیب تغییر می‌یابد.

front = (rear + 1) mod n ===➔ شرط پر بودن

front = rear ===➔ صف خالی است

برای اضافه کردن به صف حلقوی , rear یکی اضافه می‌شود و در صورتیکه rear = n – 1 باید صفر بشود. بدین منظور rear را با رابطه زیر در هر شرایطی مقداردهی می‌کنند.

rear = (rear + 1) mod n

این مسئله برای front نیز برقرار است.

front = (front + 1) mod n

برای حذف کردن

```
int delqueue ( )
{
    if (front = = rear)
    {
        C out << " صف خالی است " ;
        return 0 ;
    }
    else
    {
        front = (front + 1) mod n
        x = queue[front] ;
    }
}
```

برای اضافه کردن

```
void Addqueue (int x)
{
    rear = (rear + 1) mod n
    if (front = = rear)
        C out << " صف پر است " ;
    else
        queue[rear] = x ;
}
```

مثال :

Addqueue [50]	r = 1	0	1	2	3
	f = 0		50		
Addqueue [20]	r = 2	0	1	2	3
	f = 0		50	20	
Addqueue [30]	r = 3	0	1	2	3
	f = 0		50	20	30
delqueue ( )	r = 3	0	1	2	3
	f = 1		50	20	30
Addqueue [10]	r = 0	0	1	2	3
	f = 1	10	~~50~~	20	30