فصل سوم :
صف و پشته

- آشنایی با پشته
- آشنایی با صف
- ارزشیابی عبارات

پشته (Stack)

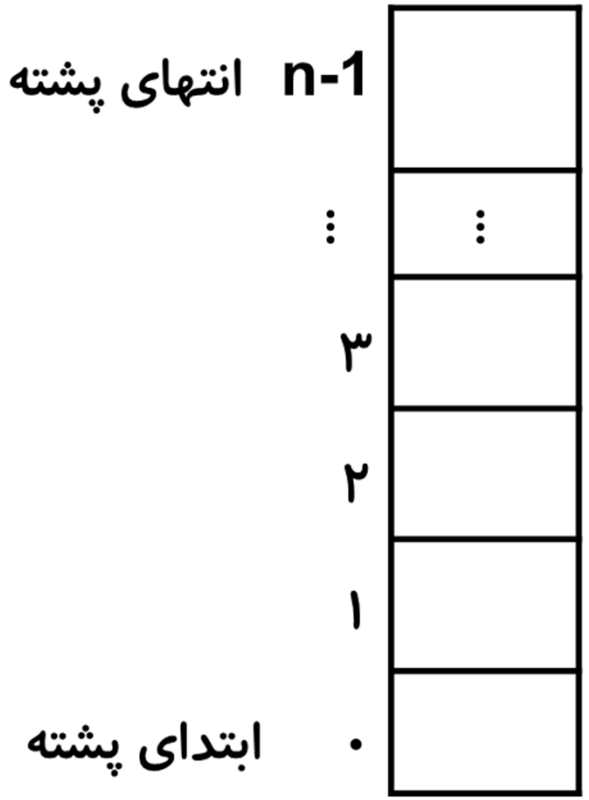
پشته و صف ، حالات خاصی از نوع داده عمومی یعنی لیست های مرتب شده ، می باشند.

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

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

پشته (Stack)

خانه های آرایه از 0 تا n-1 شماره گذاری شده و در کنار آرایه متغیری به نام TOP به عنصر بالایی پشته اشاره دارد



نوع داده `stack[n]`

دامنه تغییرات `TOP: 0 .. n`

مقدار اولیه `TOP=0`

شرط خالی بودن `TOP=0`

شرط پر بودن `TOP=n`

پشته (Stack)

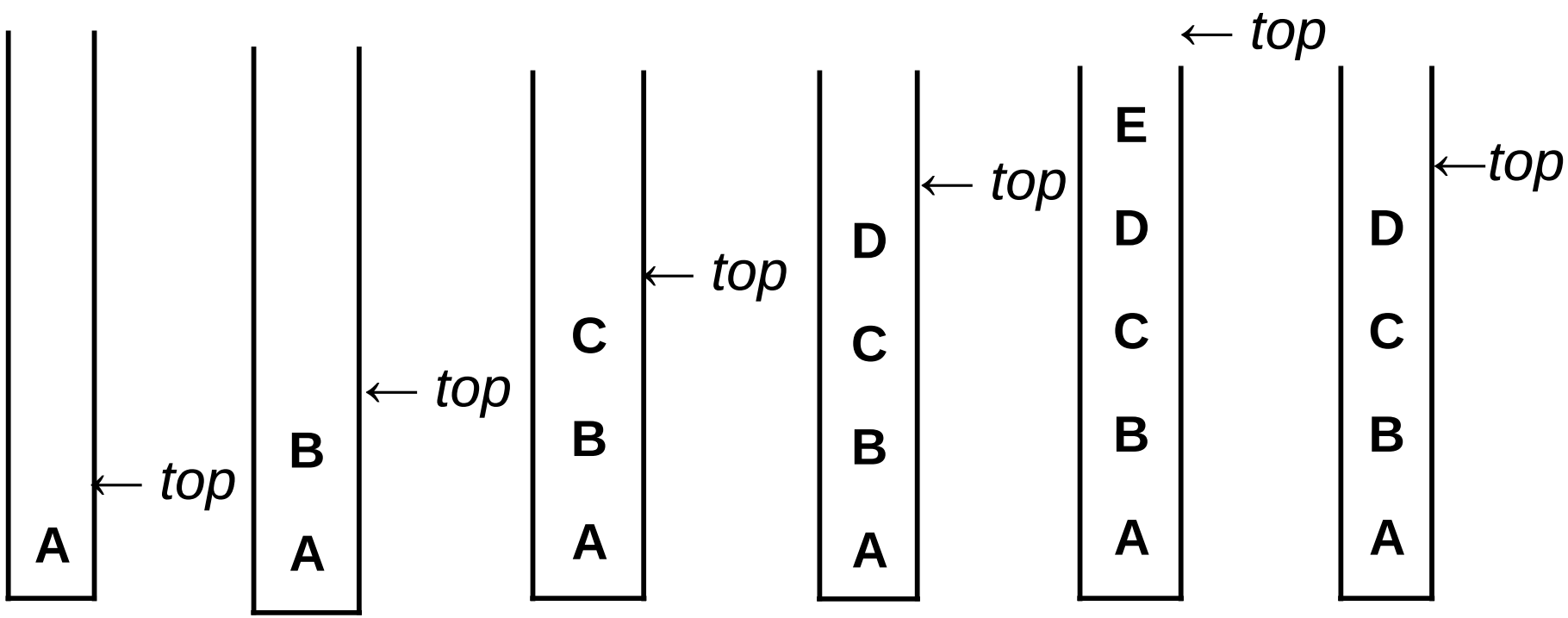
- محدودیت کار با پشته ما را ملزم می کند که اگر عناصر $A، B، C، D، E$ را به ترتیب به پشته اضافه کنیم ، E اولین عنصری خواهد بود که که از پشته حذف می گردد.
- از آنجا که آخرین عنصر وارده به پشته ، اولین عنصر حذف شده از آن می‌باشد ، پشته را به عنوان یک لیست *LIFO* (آخرین ورودی ، اولین خروجی) می شناسیم.

مهمترین توابع پشته PUSH و POP

```
void push(int x);
{
        if (top==n)
            cout << “Stack is full”;
        else
        {
            stack[top]=x;
            top++;
        }
}
```

```
int pop();
{
        if (top==0)
            cout << “Stack is empty”;
        else
        {
            top--;
            int x=stack[top];
                                }
        return x;
}
```

حذف و جایگذاری عناصر در یک پشته



صف (Queue)

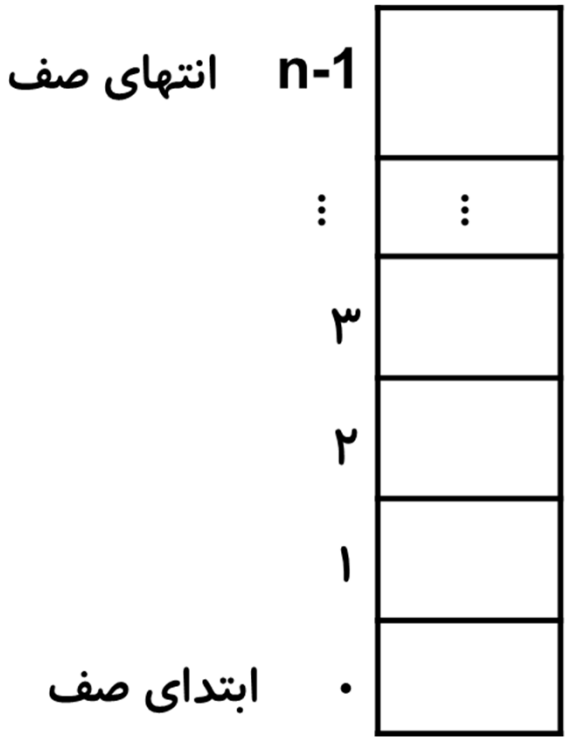
صف یک لیست مرتب است که تمامی جایگذاری آن از یک سمت و تمام حذف های ان از سمت دیگر انجام می شود.

درصف $Q = a_0, a_1, ..., a_{n-1}$ ، a_0 عنصر ابتدا (front) و a_{n-1} عنصر انتها (rear) می‌باشد و a_i در کنار a_{i+1} قرار دارد $(0 \le i < n-1)$

- از آنجا که اولین وارد شده به یک صف ، اولین عنصری است که خارج می شود ، صف را به عنوان لیست های FIFO (اولین ورودی، اولین خروجی) در نظر می گیرند.

صف خطی

خانه های آرایه از 0 تا n-1 شماره گذاری شده و در کنار آرایه به دو اشاره گر نیاز داریم؛ FRONT به عنصر قبل از عنصر ابتدایی اشاره می کند و REAR به آخرین عنصر اشاره می کند



نوع داده queue[n]

دامنه تغییرات FRONT,REAR:0.. n

مقدار اولیه FRONT=REAR=0

شرط خالی بودن FRONT=REAR

شرط پر بودن REAR=n

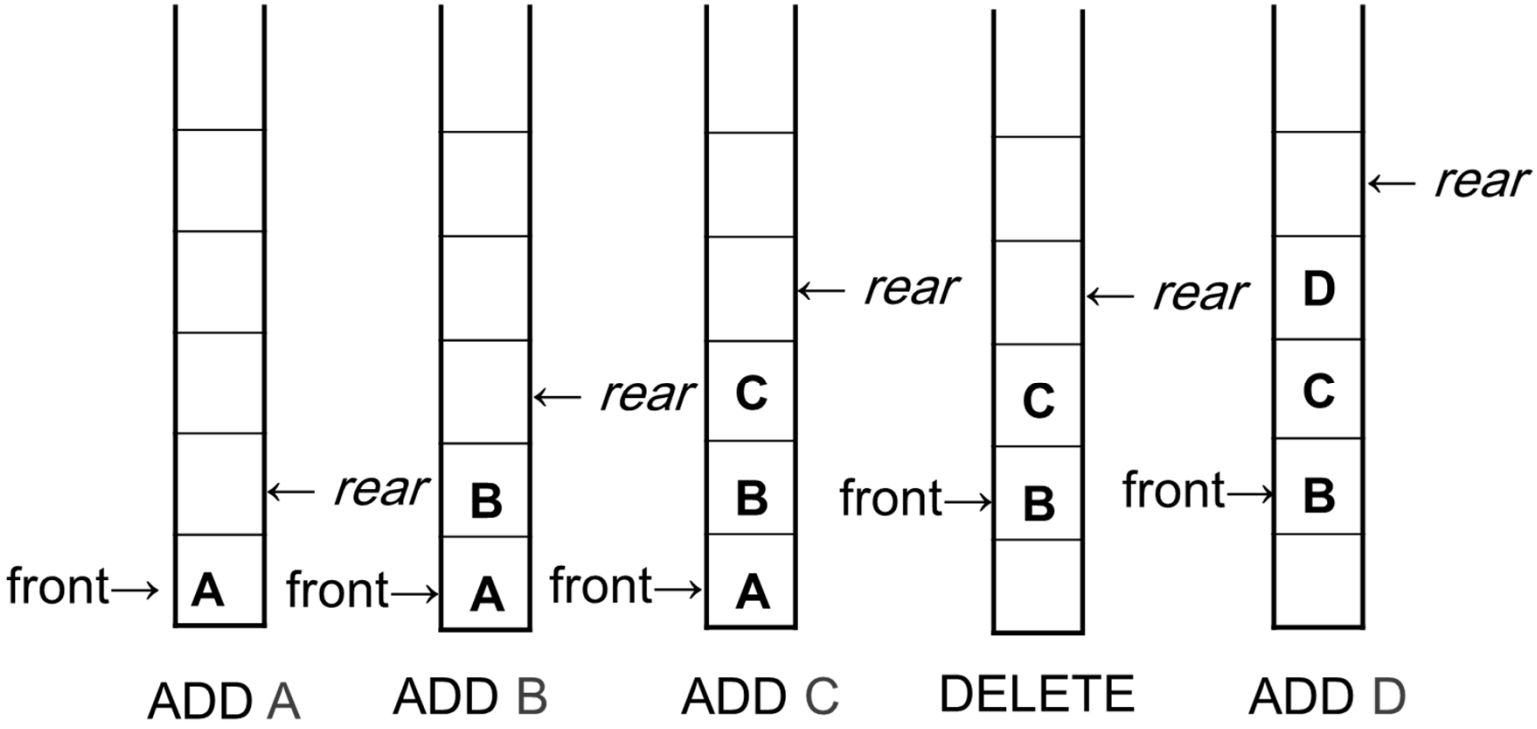
مهمترین توابع صفADD و DEL

```
void add(int x);
{
        if (rear==n)
            cout << “Queue is full”;
        else
        {
            queue[rear]=x;
            rear++;
        }
}
```

```
int del();
{
        if (front== rear)
            cout << “Queue is empty”;
        else
        {
            int x=queue[front];
            front++;
        }
        return x;
}
```

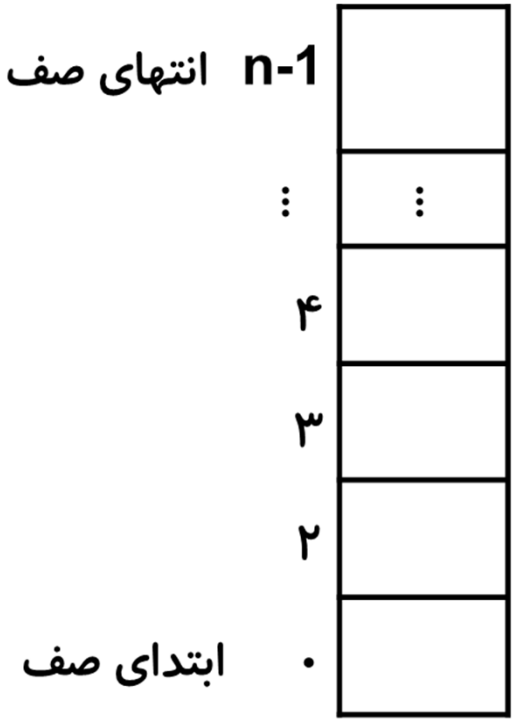
صف

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



صف چرخشی

مشکل اصلی صف خطی آن است که فقط یکبار قابل استفاده است و هنگامی که REAR به انتها می رسد نمی توان در صف چیزی را ذخیره کرد.



نوع داده queue[n-1]

دامنه تغییرات FRONT,REAR:0..n-1

مقدار اولیه FRONT=REAR=0

شرط خالی بودن FRONT=REAR

ابتدا REAR=(REAR++) % n

شرط پر بودن FRONT=REAR

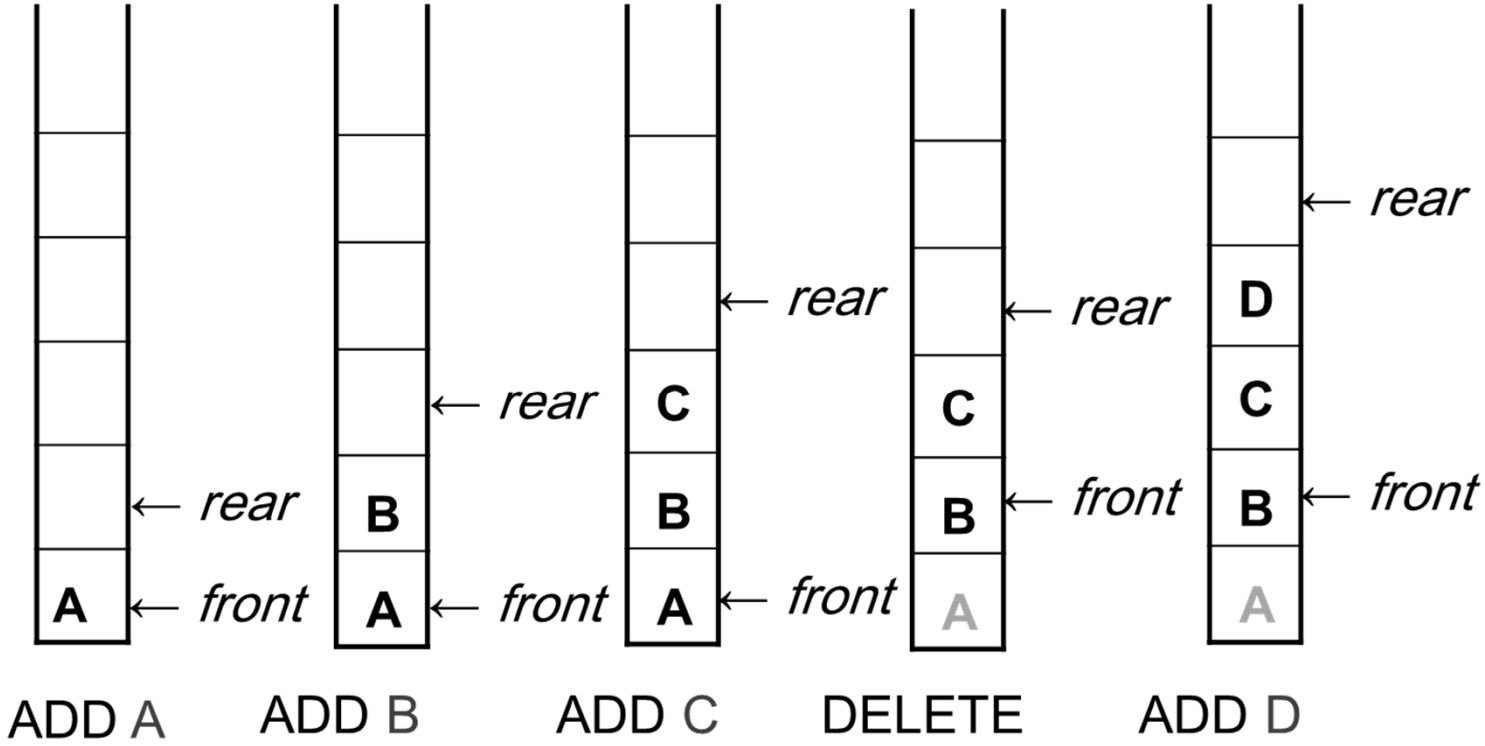
توابع صف چرخشی ADD و DEL

```
void add(int x);
{
        rear=(rear++)% n;
        if (rear==front)
            cout << “Queue is full”;
        else
        {
            queue[rear]=x;
        }
}
```

```
int del();
{
        if (front== rear)
            cout << “Queue is empty”;
        else
        {
            front =(front ++)% n;
            int x=queue[front];
        }
        return x;
}
```

درج و حذف عناصر از یک صف چرخشی

در این روش پس از حذف یک عنصر، عناصر جابجا نمی شوند بلکه تنها انـــدیس نشان دهنده آغاز صف به جلو حرکت می کند.



کاربرد Stack: ارزشیابی عبارات

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

اولویت عملگرها

جدول مقابل نشان دهنده اولویت عملگرها در زبان c می‌باشد.

Token
() [] -> .
-- ++
-- ++ ! ~ - + & * Sizeof
(type)
* / %
+ -
<< >>
>>= <<=
= = !=
&
^
|
&&
||
?:
= += -= *= %=
<<= >>= &= ^= |=

روش های نمایش عبارات ریاضی

روش های نمایش عبارت 2+3*5

- روش میان ترتیب (Infix)
  - روش استاندارد نوشتن عبارات که عملگرهای دودویی را در بین دو عملوند قرار می دهیم.

2,+,3,*,5

- روش پیش ترتیب (Prefix)

+,2,*,3,5

- روش پس ترتیب (Postfix)

2,3,5,*,+

روش پس ترتیب (postfix)

در این روش هر عملگر بعد از عملوند های مربـــوطه ظاهر می شود

مثال:

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*-

نحوه محاسبه یک عبارت Postfix

عبارات برای ارزیابی از چپ به راست پویش می شوند.

عملوندها تا مشاهده یک عملگر، داخل پشته قرار می گیرند،

سپس تعداد لازم از عملوند ها 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.

الگوریتم تبدیل infix به postfix

می توان الگوریتمی برای تبدیل یک عبارت infix به postfix به صورت زیر بیان نمود :

1- پرانتزگذاری کامل عبارات

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

3- حذف تمام پرانتزها

روش دوم تبدیل infix به postfix

با استفاده از پشته می توان عبارت infix را به postfix تبدیل نمود.

مثال: تبدیل عبارت a+b*c به نشانه گذاری postfix

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

مثال

تبدیل a*(b+c)*d به نشانه گذاری postfix

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*