ساختمان داده ها
Data Structure

فصل سوم
پشته (Stack) و صف (Queue)

تهیه و تنظیم : محمد نعیمی

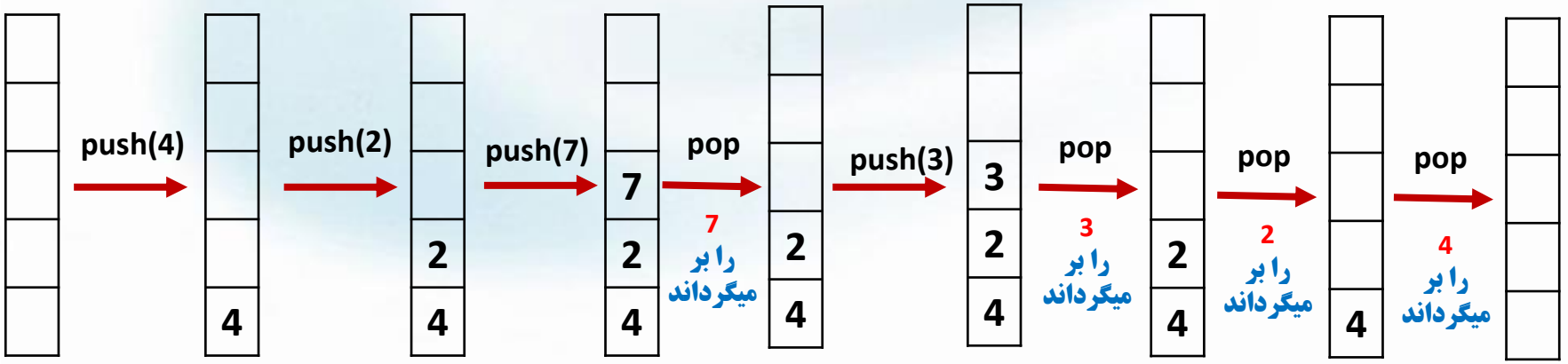
عضو هیات علمی دانشگاه آزاد اسلامی

پشته (Stack)

پشته: ساختمان داده ای است که میتوان در آن مقادری را قرار داد و از آن خارج نمود به نحوی که خروج مقادیر بر عکس ترتیب ورود آنها به پشته باشد. از این رو به آن LIFO(Last In First Out) نیز میگویند یعنی عنصری که دیرتر وارد شود زودتر خارج می شود.(مثل پارک خودرو در کوچه بن بست).

پشته ۴ تابع دارد:

push(نام پشته , مقدار)	افزودن مقدار به پشته
pop(نام پشته)	برگرداندن آخرین مقدار افزوده شده به پشته
full_stack(نام پشته)	چک نمودن اینکه آیا پشته به طور کامل پر شده است یا نه؟
empty_stack(نام پشته)	چک نمودن اینکه آیا پشته خالی است یا نه؟



پشته (Stack) – توابع پر و خالی

برای ساخت پشته می توان از رکورد شامل یک آرایه(item) و یک متغیر به نام top استفاده نمود. top به آخرین عنصر وارد شده در پشته اشاره میکند. ابتدا که پشته خالی است مقدار top برابر 1- می باشد.

از آنجایی که باید طول آرایه ابتدا مشخص باشد آن را maxsize در نظر میگیریم که یک متغیر ثابت است.

تعریف رکورد پشته

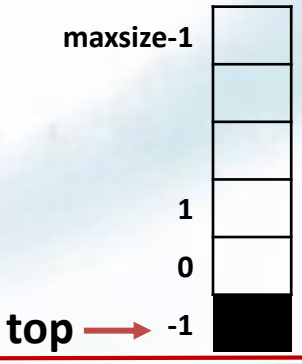
```
const int maxsize=100;
struct stack{
        type item[maxsize];
        int top;
      };
```

تعریف و مقدار دهی اولیه پشته

```
stack S;
S.top=-1;
```

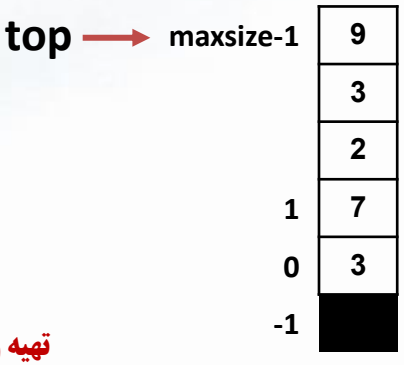
خالی بودن پشته

```
int empty_stack(stack S)
  {
    if (S.top==-1)
       return 1;
    return 0;
  }
```



پر بودن پشته

```
int full_stack(stack S)
  {
    if (S.top==maxsize-1)
       return 1;
    return 0;
  }
```



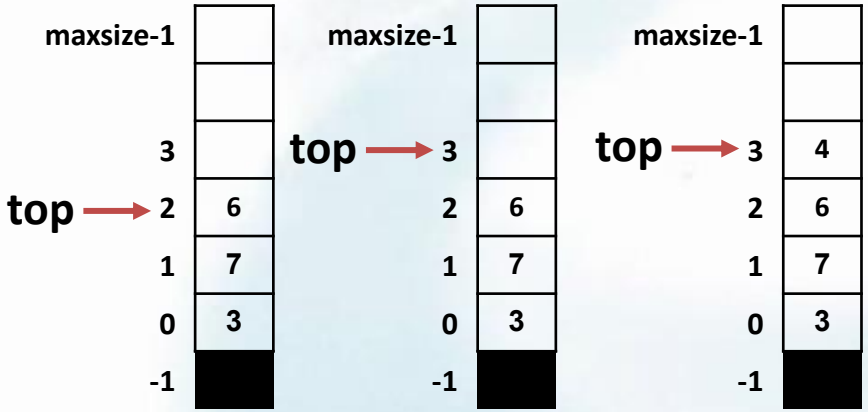
تهیه و تنظیم : محمد نعیمی

پشته (Stack) – توابع افزودن و برداشتن

```
void push(stack &S, type x)
{
  if(full_stack (S))
    cout<<"IS FULL";
  else
    S.item[++S.top]=x;
}
```

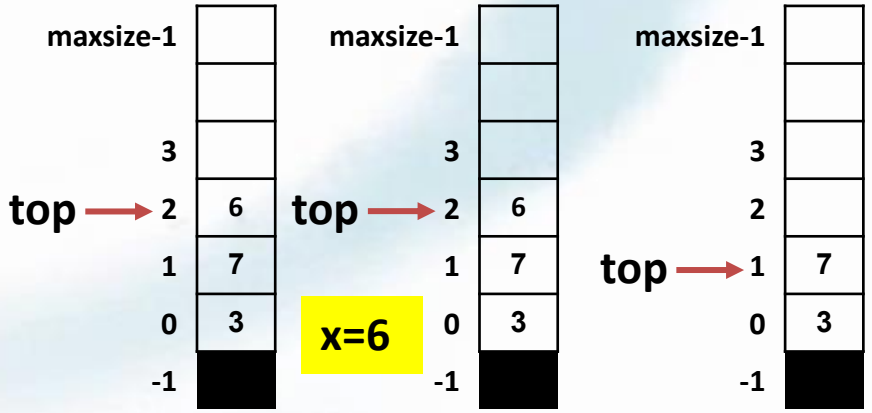
type: int, float, ...

```
{
  S.top=S.top+1;
  S.item[S.top]=x;
}
```




```
type pop(stack &S)
{
  if(empt_stack(S))
    cout<<"IS Empty";
  else
    return (S.item[S.Top--]);
}
```

```
{
  int x=S.item[S.top];
  S.top=S.top-1;
  return(x);
}
```



محتوی خانه 2 حذف نمی شود اما چون top به خانه پایین اشاره میکند
عملا خانه 2 برای ما دیده نخواهد شد

نکته برنامه نویسی: از آنجایی که میخواهیم تغییرات انجام شده در تابع روی رکورد S بر روی S اصلی نیز انجام شود باید S را به صورت by ref ارسال کنیم. در زبان C++ در خط تعریف تابع قبل از هر متغیرعلامت & بگذاریم آن متغیر by ref خواهد بود. در دو تابع صفحه قبل چون تغیری در S ایجاد نمیکنیم گذاشتن & لازم نیست.

پشته– توابع افزودن و برداشتن با اشارگر (مطالعه آزاد)

روش دیگر برای by ref نمودن متغیر تابع این است که متغیر را به صورت اشارگر (pointer) ارسال کنیم.
کافی است تابع را به صورت عادی نوشته سپس در خط تعریف قبل از نام متغیر * گذاشته و در هر کجای تابع نیز قبل از نام متغیر * بگذاریم.
تنها استثنا رکورد است که نباید در داخل تابع وقتی به فیلد آن رجوع میکنیم * بگذاریم بلکه باید . قبل فیلد را با -> جایگزین کنیم.

```
void push(stack *S, int x)
 {
    if(fullstack (*S))
      cout<<“IS FULL”;
   else
      S->item[++S->Top]=x;
 }
```

```
void pop(stack *S)
 {
   if(emptstack(*S))
     cout<<“IS Empty”;
  else
     return (S->item[S->Top--);
 }
```

هنگام فراخوان هم باید بجای نام متغیر آدرس آن را ارسال کنیم یعنی قبل از نام متغیر & بگذاریم.

```
push (&S,12)
```

فراخوان

```
cout<<pop(&S)
```

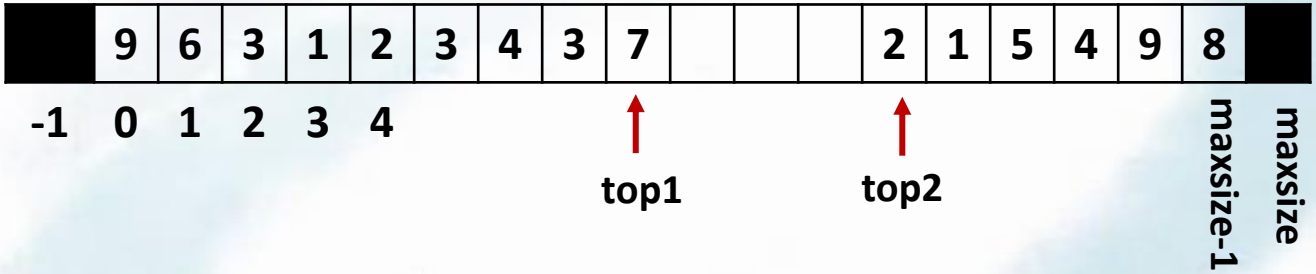
دستورات پشته به اختصار با نمایش مدلی دیگر

top میتوان بجای آخرین عنصر پشته به اولین خانه خالی پشته اشاره کند. در ابتدا اولین خانه خالی 0 می باشد
در جدول زیر مقایسه دستورات دو حالت را بررسی کرده ایم

top به بالاترین عنصر اشاره میکند. (مدل استاندارد)		top به اولین خانه خالی اشاره میکند.(مدل جدید)
top=-1	مدار دهی اولیه	top=0
top == -1	شرط خالی بودن	top == 0
top == maxsize-1	شرط پر بودن	top == maxsize
item[++top]=x	افزودن عنصر x به پشته	item[top++]=x
return(item[top--])	حذف عنصر از پشته	return(item[--top])

دستورات پشته دو طرفه

میتوان از یک آرایه برای ساخت دو پشته استفاده کرد و هر پشته از یک طرف آرایه پر شود(یکی از ابتدا و دیگری از انتها)
مزیت این روش این است که اگر هر کدام بخواهند با همین حافظه جداگانه باشند به اندازه حداکثر نصف حافظه میتوانند در یک لحظه عنصر داشته باشند اما در حالت دو طرفه در صورتی که پشته دیگر محتوای کمتری داشته باشد تا حداکثر دو برابر این حالت میتوان در یک لحظه عنصر در پشته دیگر داشته باشد.
پشته دو طرفه در بدترین حالت هر دو پشته به اندازه حالت قبل می توانند عنصر دریافت کنند.



پشته۱ از ابتدای آرایه		پشته ۲ از انتهای آرایه
top1=-1	مدار دهی اولیه	top2=maxsize
top == -1	شرط خالی بودن	top == maxsize
top1+1 == top2	شرط پر بودن	top1+1 == top2
item[++top]=x	افزودن عنصر x به پشته	item[--top]=x
return(item[top--])	حذف عنصر از پشته	return(item[top++])

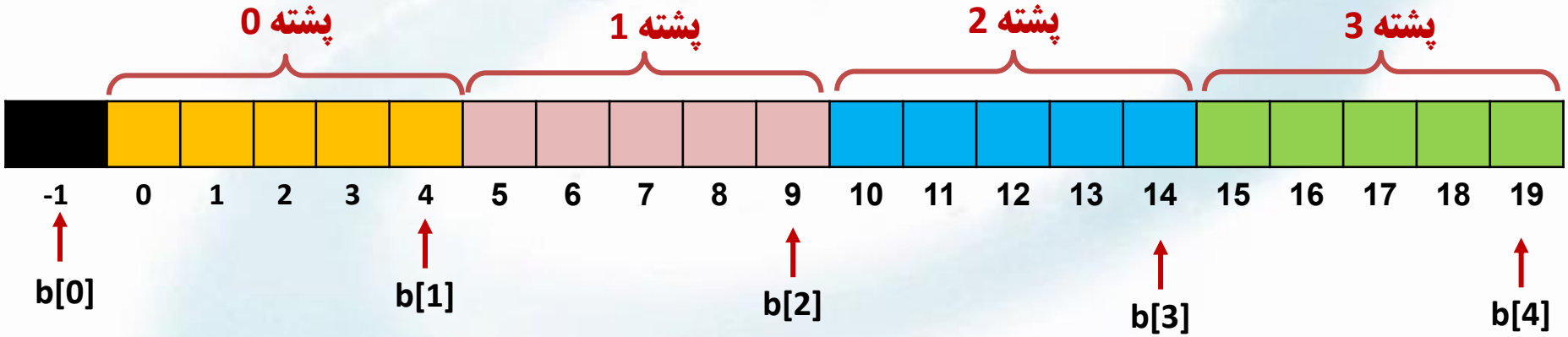
پشته چندگانه (ایجاد n پشته در آرایه ای با m خانه)

میتوان خانه های یک آرایه m عنصری را بین n پشته تقسیم کرد.
از آرایه b جهت مشخص کردن محدوده پشته ها استفاده میکنیم.
از آرایه top هم برای مشخص کردن top هر پشته استفاده می شود.

خانه های پشته i ام b[i+1] ------> b[i]+1

top[i] = b[i] = i*(m/n)-1 0 <= i < n
b[n] = n-1

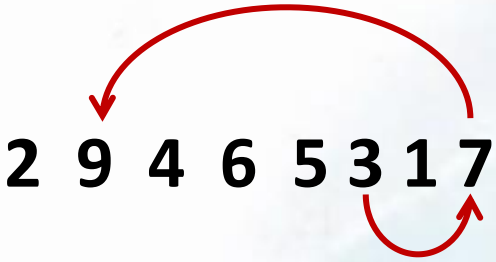
مثال زیر یک آرایه ۲۰ خانه ای را برای ساخت ۴ پشته استفاده کرده است(هر پشته ۵ خانه)



مدار دهی اولیه پشته i ام	top[i]=b[i]
شرط خالی بودن پشته i ام	top[i] == b[i]
شرط پر بودن پشته i ام	top[i] == b[i+1]
افزودن عنصر x به پشته i ام	item[++top[i]]=x
حذف عنصر از پشته i ام	return(item[top[i]--])

دنباله های مجاز خروجی یک پشته

اگر اعداد 1 تا n به ترتیب صعودی به عنوان ورودی پشته در نظر گرفته شوند و با ترکیب مختلف از push و pop خروجی های متفاوتی از پشته امکان پذیر خواهد بود.
تنها یک نوع خروجی با هر ترکیب push و pop قابل تولید نیست. هر خروجی که در آن بعد از عددی دو عدد کوچکتر از آن به صورت صعودی بیایید مجاز نیست. (بعد از عددی یک عدد بزرگتر از آن بیایید و قبل از هر دو عددی بزرگتر از آنها)



ترتیب خروجی از چپ به راست

در مثال روبرو تناقض بسیار است که فقط یکی بیان شده تا نشان دهیم لازم نیست الزاما عدد ها پشت سر هم یا در مکان مجاور هم باشند

کاربرد های پشته:

از پشته جهت مدیریت فراخوان توابع استفاده می شود.
اگر در خط 15 تابع main برنامه، تابع fun_1 فراخوان شود و تابع fun_1 در خط 7 خود تابع fun_2 را فراخوان کند و تابع fun_2 در خط 29 خود تابع fun_3 را فراخوان کند هنگامی که تابع fun_3 پایان بپذیرد باید نتیجه را به کدام تابع برگرداند؟

قطعا باید به خط 29 تابع fun_2 برویم لذا تابعی که دیرتر فراخوان شده نتیجه به آن بر میگردد.

3	
2	fun_2 - line 29 - other data
1	fun_1 - line 7 - other data
0	main - line 15 - other data

جهت مدیریت این موضوع هنگام فراخوان هر تابع اطلاعات مربوط به تابع فعلی در یک پشته ذخیره میشود و هنگام پایان تابع آخرین تابعی که در پشته قرار دارد جهت دریاقت خروجی تابع و ادامه کار از خطی که در آن تابع را فراخوان کرده فعال می شود.
در توابع بازگشتی نیز به همین شکل مشخص می شود که پاسخ هر تابع به کدام مرحله برگردد

ارزیابی عبارات:

برای محاسبه عبارت 2+3*4 ، اگر ابتدا ضرب را انجام دهیم بعد جمع نتیجه با زمانی که اول جمع را انجام دهیم بعد ضرب متفاوت خواهد شد.

این موضوع **ابهام عبارت** نام دارم.

این ابهام ناشی از شیوه فرمول نویسی می باشد. از آنجایی که بین عملوند ها از عملگر استفاده میکنیم عدد 3 از یک طرف مجاور + می باشد و از سمت دیگر مجاور * می باشد. به این فرم نوشت عبارت روش infixed میگوییم.

- فرمول ها بر اساس محل قرار گیری عملگر به سه فرم زیر قابل نمایش هستند

prefixed (پیشوندی): عملگر سمت چپ (قبل از) دو عملوند ظاهر می شود *AB

infixed (میانوندی): عملگر بین دو عملوند ظاهر می شود A*B (فرمی که معمولا همه استفاده میکنیم)

postfixed (پسوندی): عملگر سمت راست (بعد از) دو عملوند ظاهر می شود AB*

در روش prefixed و postfiexd ابهام وجود نخواهد داشت

بالاترین اولویت	^ توان
	* / %
پایین ترین اولویت	+ -

برای رفع ابهام عبارات infixed لازم است اولویت عملگر ها را مشخص کنیم یا از پرانتز گذاری استفاده نماییم.

در c++ توان نداریم اما از نماد مقابل در مثالها استفاده میکنیم

روش تبدیل فرم infixed به prefixed و postfixed

برای تبدیل فرم infixed (میانوندی) به دو فرم دیگر به شکل زیر عمل میکنیم.

- برای تبدیل فرم infixed به postfixed (پسوندی) لازم است ابتدا فرمول را پرانتز گذاری کامل کرده سپس هر عملگر را به محل پرانتز بسته آن منتقل میکنیم. برای پرانتز گذاری درست میتوان:
  - ابتدا از چپ به راست تمام توانها را پرانتز گذاری کرده و عملگر توان را بالای پرانتز بسته مربوطه میگذاریم
  - مرحله دوم از چپ به راست تمام ضرب و تقسیم ها را پرانتز گذاری کرده و عملگر آن را بالای پرانتز بسته مربوطه میگذاریم
  - مرحله سوم از چپ به راست تمام جمع و تفریق ها را پرانتز گذاری کرده و عملگر آن را بالای پرانتز بسته مربوطه میگذاریم
  - در مرحله آخر عملوند ها را نیز بالای خود آنها نوشته و عبارت تولید شده در بالا همان عبارت postfixed (پسوندی) است.

دقت کنید جایی که خود عبارت پرانتز دارد باید از پرانتز آن استفاده کنید و کل پرانتز عملا یک عملوند برای عملگر های بیرونی آن به حساب می آید. (ابتدا داخل پرانتز حساب میشود بعد نتیجه با عملگر های بیرونی اعمال می شود)

در مراحل فوق پرانتز بسته ها (عموما بعد از عملوند یا پرانتز بسته و قبل از عملگر یا انتهای عبارت قرار میگیرد و پرانتز باز ها) عموما بعد از عملگر یا ابتدای عبارت و قبل از عملوند یا پرانتز باز قرار میگیرد

برای ساخت prefixed (پیشوندی) کافی است عملگر به محل پرانتز باز منتقل شود

مثال ۱: تبدیل infixed به postfixed

A - B * K / T ^ D + F * P

A - B * K / (T ^ D)^ + F * P

مرحله توان از چپ به راست

A - (B * K)* / (T ^ D)^ + F * P

A - ((B * K)* / (T ^ D)^)/ + F * P

A - ((B * K)* / (T ^ D)^)/ + (F * P)*

مرحله ضرب و تقسیم از چپ به راست

(A - ((B * K)* / (T ^ D)^)/)- + (F * P)*

((A - ((B * K)* / (T ^ D)^)/)- + (F * P)*)+

مرحله جمع و تفریق از چپ به راست

A B K * T D ^ / - F P * +

postfixed : ABK*TD^/-FP*+

مثال ۲: تبدیل infixed به prefixed

A - B * K / T ^ D + F * P

A - B * K / ^(T ^ D) + F * P — مرحله توان از چپ به راست

A - *(B * K) / ^(T ^ D) + F * P

A - /*((B * K) / ^(T ^ D)) + F * P

A - /*((B * K) / ^(T ^ D)) + *(F * P) — مرحله ضرب و تقسیم از چپ به راست

-(A - /*((B * K) / ^(T ^ D))) + *(F * P)

+-((A - /*((B * K) / ^(T ^ D))) + *(F * P)) — مرحله جمع و تفریق از چپ به راست

+-A /*B K ^T D *F P

prefixed : +-A/*BK^TD*FP

مثال بیشتر تبدیل فرم عبارات از infixed به دو فرم دیگر

پرانتز مشکی جزیی از عبارت اصلی است

in: M * B - F / K ^ (T + C * E) ^ N

M * B - F / ((K ^ (T + C * E)) ^ N)

(M * B) - (F / ((K ^ (T + (C * E))) ^ N))

((M * B) - (F / ((K ^ (T + (C * E))) ^ N)))

pre: -*MB/^^K+C*EN post: MB*FKTCE*+^N^/-

in: M - B + F / K * T ^ C / E + N

\- / ^* /+ +

(((M - B) + (((F / K) * (T ^ C)) / E)) + N)

++- /*/ ^

pre: ++-MB/*/FK^TCEN post:MB-FK/TC^*E/+N+

الگوریتم تبدیل عبارت infixed به postfixed با استفاده از پشته

عبارت را از چپ به راست میخوانیم:

- اگر عملوند خوانده شد، آن را چاپ میکنیم
- اگر عملگر خوانده شد :
  - اگر پشته خالی بود عملگر را به پشته اضافه میکنیم.
  - اگر پشته خالی نبود، عملگر جدید را با عملگر بالای پشته مقایسه میکنیم:
    - اگر اولویت عملگر جدید بیشتر بود آن را به پشته اضافه میکنیم
    - وگرنه آنقدر از پشته عملگر خارج نموده و چاپ میکنیم که یا پشته خالی شود یا به پرانتز باز برسیم یا اولویت عملگر جدید بالاتر از عملگر بالای پشته شودکه بعد از آن عملگر جدید را به پشته اضافه میکنیم.
- اگر پرانتز باز خوانده شد، آن را بی هیچ قید و شرطی در پشته می گذاریم.
- اگر پرانتز بسته خوانده شد، آنقدر عملگر از پشته برداشته و چاپ میکنیم تا یک پرانتز باز خارج شود. این پرانتز باز و بسته را دور می اندازیم و چاپ نمیکنیم.
- اگر به انتهای عبارت رسیدیم هرچه در پشته داریم خارج نموده و چاپ میکنیم

مثال :(تبدیل عبارت infixed به postfixed با استفاده از پشته)

عبارت	a	+	b	*	c	/	(	k	*	(	d	+	e	-	f	)	)	-	s
ترتیب خواندن	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

	محتوای پشته از چپ به راست	
2	+	
4	+ *	
6	+ /	* خروج
7	+ / (	
9	+ / (*	
10	+ / (* (	
12	+ / (* (+	
14	+ / (* (-	+ خروج
16	+ / (*	- خروج
17	+ /	* خروج
18	-	/ خروج + خروج
end		- خروج

1	3	5	6	8	11	13	14	15	16	17	18	19	end
a	b	c	*	k	d	e	+	f	-	*	/+	s	-

الگوریتم محاسبه مقدار یک عبارت postfixed:

عبارت را از چپ به راست میخوانیم:

- اگر عملوند خوانده شد، آن در پشته می گذاریم
- اگر عملگر خوانده شد دو عنصر از پشته خارج میکنیم (عنصر بالایی پشته عملوند دوم و عنصر بعدی عملوند اول) و عملگر را روی آنها اعمال نموده و نتیجه را در پشته قرا می دهیم.
- وقتی به انتهای عبارت رسیدیم محتوای پشته حاصل عبارت می باشد

مثال: post:5 3 4 * + 2 4 ^ 2 / - in: (5+(3*4))-((2^4)/2)

5	3	4	*	+	2	4	^	2	/	-
			3*4	5+12			2^4		16/2	17-8

		4				4		2		
	3	3	12		2	2	16	16	8	
5	5	5	5	17	17	17	17	17	17	9

صف (Queue)

صف: ساختمان داده ای است که میتوان در آن مقادری را قرار داد و از آن خارج نمود به نحوی که خروج مقادیر دقیقا به ترتیب ورود آنها به صف باشد. از این رو به آن FIFO(First In First Out) نیز میگویند یعنی عنصری که زودتر وارد شود زودتر خارج می شود.(مثل تمامی صف های موجود در جامعه).

ایده اول:

- ساده ترین ایده چیزی شبیه صف در بانک است که محل باجه (ابتدای صف) معلوم و ثابت است.
- ابتدای صف همیشه خانه 0 می باشد و کافی است یک متغیر(r) برای تعیین مکان آخرین عنصر صف در نظر بگیریم.
- با خروج عنصر، سایر عناصر یک خانه به سمت خانه 0 خواهند رفت. پیچیدگی عملیات شیفت n عنصر o(n) می باشد.

عیب:

پیچیدگی حذف در این روش o(n) می باشد

ایده دوم:

- برای حل مشکل شیفت عناصر صف، میتوان یک متغیر(f) نیز برای تعیین مکان عنصر اول صف گذاشت که با خروج هر عنصر مقدار f یک واحد اضافه می شود. پیچیدگی در این حالت o(1) می باشد.

عیب:

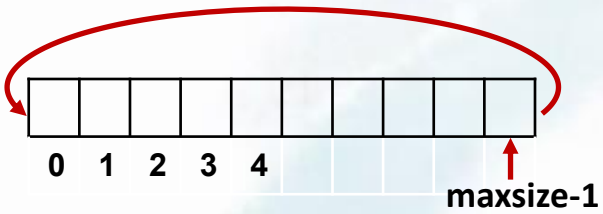
چون ابتدای صف با تغییر مقدار f حرکت میکند لذا خانه های قبل از f خالی می باشند و با رسیدن r به انتهای آرایه پیغام "صف پر است" چاپ خواهد شد در حالی که تعدادی خانه قبل از f وجود دارد که خالی است.

در این روش عملا خانه ها یکبار مصرف خواهند بود.

صف (Queue)– ایده کامل و دقیق

ایده سوم (کامل و دقیق):

- **این روش با اصلاح مشکل خانه های خالی قبل از f صف را طراحی میکند.**
- **در این روش صف به صورت حلقوی میباشد. یعنی خانه بعد از maxsize-1 می شود خانه 0. به عبارت دیگر f و r هنگامی که حرکت میکنند یک واحد به آنها اضافه می شود، اما وقتی درخانه maxsize-1 باشند به خانه 0 می روند.**



برای این منظور فرمول زیر پیشنهاد می شود(r نیز به همین شکل است)

$$f=(f+1)\%maxsize$$

از آنجایی که باقی مانده اعداد کوچکتر از maxsize بر maxsize خود آن عدد می شود لذا در حالت عادی مقدار (f+1)%maxsize معادل f+1 خواهد شد (یعنی خانه بعد آرایه)

در حالتی که عدد برابر maxsize باشد باقی مانده آن بر maxsize برابر 0 خواهد شد یعنی وقتی f=maxsize-1 باشد f+1=maxsize خواهد بود لذا (f+1)%maxsize می شود 0 یعنی خانه بعد از maxsize-1 خانه 0 خواهد شد.

$$(f+1)\%mazsize = \begin{cases} f+1 & f+1 < maxsize \\ 0 & f+1 = maxsize \end{cases}$$

صف (Queue)–تعریف رکورد و مقدار اولیه

addq(نام صف , مقدار)	افزودن مقدار به صف
delq(نام صف)	برگرداندن آخرین مقدار افزوده شده به صف
full_queue(نام صف)	چک نمودن اینکه آیا صف به طور کامل پر شده است یا نه؟
empty_queue(نام صف)	چک نمودن اینکه آیا صف خالی است یا نه؟

تعریف و مقدار دهی اولیه صف

```
const int maxsize=100;
struct queue{
        type item[maxsize];
        int f,r;
};
queue Q;
Q.r=Q.f=0;
```

تعریف رکورد صف

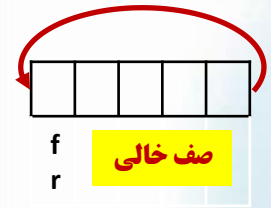
در صف حلقوی f به خانه قبل از اولین عنصر صف و r به خانه عنصر آخر صف اشاره می کند . مشکلی که در صف حلقوی بوجود خواهد آمد این است که شرط پر و خالی بودن یکسان خواهد شد (f==r).

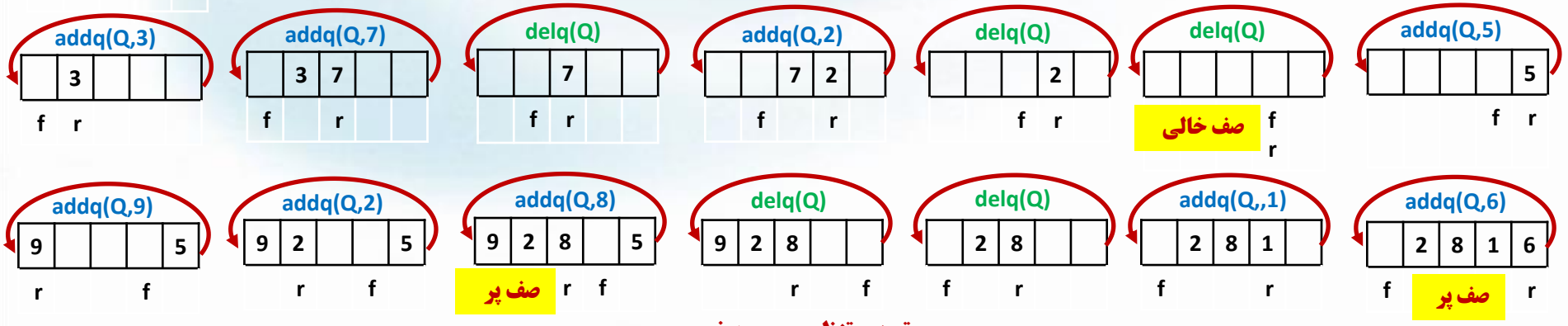
برای رفع این مشکل دو راه حل وجود دارد:

- در متغیری آخرین عملیات شامل افزودن یا برداشتن ثبت شود.(ما از این روش استفاده نمیکنیم)
- یک خانه را همیشه خالی نگه داریم یعنی هنگامی که r به خانه قبل (حلقوی) از f رسید اعلام کنیم صف پر است.

در این حالت f==r شرط خالی بودن خواهد بود و (r+1)%maxsize == f شرط پر بودن صف.

مقدار دهی اولیه برای r و f کافی است یک اندیس یکسان مثل 0 باشد (مقدار 1- مجاز نیست)





توابع صف (Queue)

پر بودن پشته

```
int full_queue(Queue Q)
  {
    if ((Q.r+1)%maxsize==Q.f)
        return 1;
    return 0;
  }
```

خالی بودن پشته

```
int empty_queue(queue Q)
 {
   if (Q.f==Q.r)
       return 1;
   return 0;
 }
```

افزودن عنصر به صف

نکته:
در افزودن عنصر به صف مثل صف بانک ما فقط با آخر صف کار داریم یعنی Q.r

```
void addq(queue &Q, int x)
{
   if(full_queue (Q))
     cout<<"IS FULL";
  else
    {
       Q.r=(Q.r+1)%maxsize;
       Q.item[Q.r]=x;
    }
}
```

برداشتن عنصر از صف

نکته:
در برداشتن عنصر از صف مثل صف بانک ما فقط به اول صف کار داریم یعنی Q.f

```
void delq(Queue &Q)
{
  if(empt_queue(Q))
    cout<<"IS Empty";
  else
    {
      Q.f=(Q.f+1)%maxsize;
      return (Q.item[Q.f]);
    }
}
```