به نام خدا

ساختمان داده

مرکز علمی کاربردی شهرداری بیارجمند

مدرس:محمد ابراهیم باقری

ساختمان داده

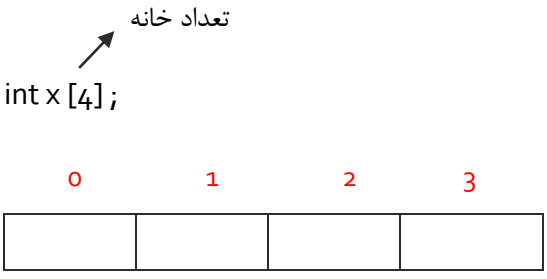
جلسه اول

سرفصل ها:

- آرایه و عملیات مربوطه
- ماتریس
- صف
- پشته
- توابع بازگشتی
- عبارات محاسباتی
- لیست پیوندی
- گراف
- مرتب سازی

آرایه

ساختمان داده ایست که از تعدادی خانه پشت سرهم و هم نوع تشکیل شده است.



مثال. Int x [A] چند خانه دارد ؟

جواب : 65 خانه

نکته : در c# این قطعه کد عمل نمی کند و خطا دارد .

نکته : برای بدست آوردن تعداد خانه n تا m اولی – دومی + 1 می کنیم

نمونه: ‘A’ تا ‘Z’

جواب: 26=1 + 65 -90

حافظه مصرفی

فرمول حافظه مصرفی به شرح ذیل است :

حافظه مصرفی = تعداد خانه * تعداد بایت

نکته : در ویندوز تعداد بایت 4 است اما در کنکور تعداد بیت را 8 در نظر می گیریم.

مقدار مصرفی آرایه

روش های محاسبه مقدار محاسبه آرایه به شرح ذیل است:

- در زمان تعریف
- مقداردهی خانه به خانه
- از ورودی
- مقدار دهی بصورت خودکار

• با مقادیر تصادفی

روش اول : در زمان تعریف

int x [4]={3 , 2 , 5 , 9};

نمایش =

3	2	5	9

int x [4] = {3 , 2}

نمایش =

3	2	0	0

در خانه های خالی مقدار پیش فرض را از نوع مورد نظر می گذاریم.

مقادیر پیش فرض عبارت اند از :

صحیح	0
اعشاری	0.0
کاراکتری	\0
اشاره گر	NUPP

انواع خطاها عبارت اند از:

ترجمه

اجرا

منطقی

خطای زمان ترجمه مانند :

int[4]= {3 ,2 ,8, 5 , 7}

خطای منطقی مانند :

بطور مثال در C اگر صورت صحیح و مخرج صحیح باشد جواب هم صحیح است (float 15/4 = 3) اما شما 3.75 می خواهید که برنامه نمی دهد پس باید یکی از بین صورت و مخرج را اعشاری کنید تا خطای منطقی رخ ندهد (float 14/4.0 = 3.75) .

خطای اجرا مانند : x=y/0

روش دوم : مقدار دهی خانه به خانه

مزیت این روش نسبت به روش قبلی این است که در روش قبل اگر می خواستید بطور مثال خانه چهارم را مقداردهی کنید ، ابتدا باید تمامی خانه های قبل را اول مقدار دهی می کردید که در این روش نیاز نیست.

```
int x [4];
x[3] = 2;
x[0] = 1;
```

روش سوم: مقداردهی از ورودی

```
for (int i =0 ; i>4 ; i++)
cin >>x[i];
```

C#

```
Console
int.parse (console.readline())
console.writeline()
from
int.parse (textbox.text())
```

C

بگیر ↓ آدرس ↓

```
scanf("%d",& x[i])
```

printf (“%d”,x[i]) خروجی

C++

```
cin >> x[i];
cout << x[i];
```

روش چهارم: مقداردهی بصورت خودکار

```
for (int i =0 ; i>4 ; i++)
x[i]=i
```

0	1	2	3

سوال کنکوری=

```
for (int i =0 ; i>4 ; i++)
x[i] = i++;
```

جواب :

0		2	

نکته : ++i دارای اولویت زیاد می باشد و i++ دارای اولویت کم .

روش پنجم: با مقادیر تصادفی

```
for (int i =0 ; i>4 ; i++)
x[i]=random(20);
```

انواع توابع تصادفی :

1- تابعی که پارامتر نمی گرید.
2- تابعی که یک پارامتر می گیرد — rnd (n); عددی بین 0 تا n می دهد
3- تابعی که دو پارامتر می گرد — r.next(m,n); عددی بین m<=random <n

کاربرد عدد تصادفی ؟

زمانی که می خواهیم یک برنامه را چک کنیم تا ببینیم درست عمل می کند یا نه از عدد تصادفی استفاده می کنیم .

سوال . قطعه کدی بنویسید که نمره یک درس را بدهد و اعشاری هم بدهد .

جواب :

```
x =random(21)+random(100)/100.0
if (x>20)
x=20;
```

عملیات روی آرایه

انواع عملیات روی آرایه :

- جستجو (خطی ،دودویی ، شاخص(index) ، درهم سازی (hashing)
- مرتب سازی

جستجوی خطی

جستجو از ابتدای آرایه شروع می شود و نیاز به مرتب کردن آرایه نیست و هرجا که عنصر مورد نظر پیدا شود .

```
cout << "pz enter num";
cin >>num ;
if (x[i] == num)
{
cout << "found";
flag=1;
break;
}
if (flag==0)
```

cout << “Not found”;

نکته کنکوری : تعداد مقایسه در جستجوی خطی در آرایه nعنصری:

حداقل 1

حداکثر n

میانگین $\frac{1+n}{2} = \frac{n}{2}$

جستجوی دودویی

عمل جستجو از وسط شروع می شود ، اگر پیدا شود جستجو تمام است ، اگر از آن کوچکتر باشد وسط به پایین و اگر بزرگتر باشد وسط به بالا و تکرار.

پیدا کردن 11؟

2	5	7	9	11	13	15

مقایسه 1 مقایسه 3 مقایسه 2

نکته : خانه اول و آخر بیشترین مقایسه را دارند .

نکته : کمترین مقایسه ، یک مقایسه است .

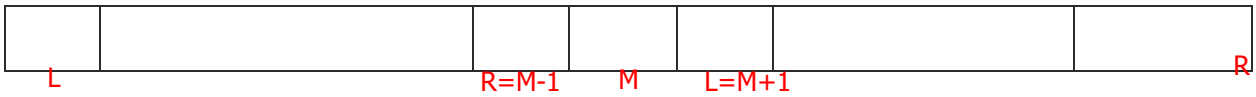
نکته : بیشترین مقایسه را می توان با فرمول $(log_2^n) + 1$

بدست می آید .

سوال : در یک آرایه مرتب شده $80*10^6$ حد اکثر تعداد مقایسه در جستجوی خطی و دودویی کدام است

خطی n= 80000000

دودویی= $log_2^{80*10^6} + 1 = 27$



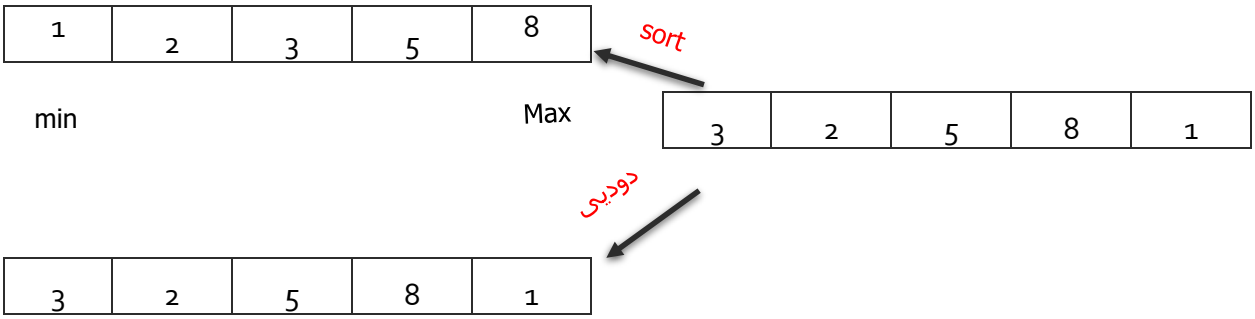
```
flag =0;
l =0;
r=n-1;
while (l<=r)
{
m=l+r/2;
if (x[m]==num)
{
cout << "found";
flag=1;
break;
}
else if (x[m] <num)
r =m-1;
if (x[m] >num)
l=m+1;
}
if (flag ==0)
cout << "not found";
```

پایان جلسه اول

ساختمان داده

جلسه دوم

الگوریتم پیدا کردن مینیموم و ماکزیموم



برای پیدا کردن max و min می توان از دو روش جستجوی دودویی و مرتب سازی استفاده کرد.

تعداد مقایسه دریافتی از آرایه n عنصری :

حداقل n-1

حداکثر n-1

min

```
max = x[o];
for (i=1; i<n ;i++)
if (x[i] <min)
min=x[i];
cout << min;
```

آرایه دو بعدی

ساختمان داده ایست که از تعدادی سطر و ستون تشکیل شده است .

Int x [3] [4] ;

تعداد عنصر : 3*4=12

میزان حافظه : 12 * حافظه مورد نظر

پر کردن آرایه دو بعدی

این روش نیازمند دو حلقه for است ، یکی برای سطر و یکی برای ستون

```
cout int row =3;
cout int col=4;
int x[row] [col];
for (int i=o ;i<row;i++)
for (int j =o;j<col;j++)
x [i] [j] = random (21);
```

نمایش محتوای آرایه دوبعدی

```
for (int i=o ;i<row;i++)
{
for (int j =o;j<col;j++)
{
cout <<x[i] [j] <<'','';
```

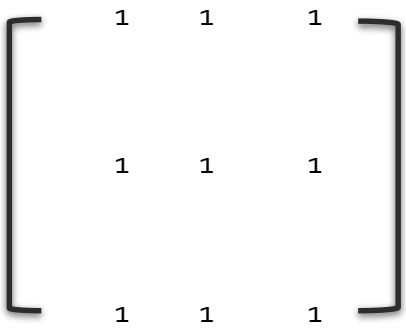
```
}
```

این دستور باعث میشه خروجی ما شکل ماتریسی بگیرد

```
cout << 'in';
}
```

ماتریس

ساختمان داده ایست که از تعدادی سطر و ستون تشکیل شده است و با آرایه دو بعدی پیاده سازی می شود .



عملیات های ماتریس :

- جمع
- تفریق
- ضرب
- تقسیم

جمع دو ماتریس

در جمع دو ماتریس باید تعداد عناصر هر دو ماتریس یکسان باشد .

در جمع دو ماتریس $A_{mn} + B_{mn}$ نیاز به mn عمل جمع است .

در جمع ماتریس نیاز به دو حلقه است .

```
cout int row =3;
cout int col =4;
int A [row] [col];
int B [row] [col];
int C [row] [col];
for (int i=0 ;i<row;i++)
for (int j =0;j<col;j++)
c[i] [j] = A [i] [j] + B [i] [j]
```

ضرب دو ماتریس

تعداد ستون ماتریس اول باید با تعداد سطر ماتریس دوم برابر باشد .

$$A_{mn} + B_{np} = c_{mp}$$

در ضرب دو ماتریس $A_{mn} + B_{np}$ تعدادضرب mnp است.

در ضرب دو ماتریس $A_{mn} + B_{np}$ تعداد عمل جمع m(n-1)p است.

سوال : در ضرب دو ماتریس A7*8 و B8*5 چند عمل ضرب نیاز است ؟

جواب : 7 * 8 * 5 = 280

```
const int m =2;
const int n =4;
const int p =3;
int A [m] [n];
int B [n] [p];
int C [m] [p];
for (int i=0 ;i<m;i++)
for (int j=0 ;j<n;j++)
```

```
c[i] [j] =0;
for (int k=0 ; k<m; k++)
c[i] [j] = C[i] [j] + a [i] [k] * B [k] [j];
```

انواع ماتریس

- ماتریس خلوت
- ماتریس سه قطری
- ماتریس مثلثی

ماتریس خلوت

ماترسیست که بشتر عناصر آن 0 است .

$$\begin{bmatrix} 4 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 9 & 0 & 0 & 0 \\ 7 & 4 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

40 * 100

کل عناصر=500*1000

عناصر غیر صفر7 =

عنصر صفر4993 =

سطر	ستون	مقدار
0	0	3
0	2	8
1	1	6
1	3	9
2	0	7
1	1	5
2	2	4

نحوه پیاده سازی :

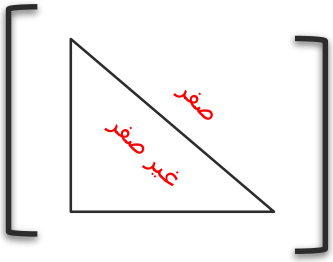
- روش عادی int [50] [100]
- روش مطلوب

3n * n = تعداد خانه (n تعداد عنصر غیر 0 است)

ماتریس مثلثی

ماتریسی است مربعی که عناصر روی قطر اصلی و پایینی آن غیر صفر و باقی صفر است .

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 6 & 5 & 8 & 0 \end{bmatrix}$$



سوال : در یک ماتریس پایین مثلثی تعداد عناصر صفر و غیر صفر چقدر است .

غیر صفر : $1+2 + \ldots + n=n/2\ (n+1)$

تعداد صفر :$n/2(n-1)$

ماتریس بالا مثلثی

ماتریسی است مربعی که عناصر روی قطر اصلی و پایینی آن غیر صفر و باقی صفر است.

نکته : فرول ها یکسان هستند .

پایان جلسه دوم

ساختمان داده

جلسه سوم

روش پیاده سازی ماتریس پایین مثلثی

$$\begin{bmatrix} 3 & 0 & 0 & 0 \\ 2 & 5 & 0 & 0 \\ 6 & 8 & 9 & 0 \\ 10 & 13 & 6 & 15 \end{bmatrix}$$

روش عادی :

```
int m [9] [9] ;
```

روش مطلوب :

0	1	2	3	4	5	6	7	8	9
3	2	5	6	8	9	10	13	4	15

ماتریس سه قطری

ماتریس است که عناصر قطر اصلی بالایی و پایینی غیر صفر و مابقی صفر است .

$$\begin{bmatrix} 3 & 1 & 0 & 0 \\ 11 & 2 & 7 & 0 \\ 0 & 13 & 5 & 6 \\ 0 & 0 & 15 & 8 \end{bmatrix}$$

سوال : در یک ماتریس 3 قطری 20 *20 تعداد عناصر 0 و غیر صفر چقدر است ؟

کل عناصر : 20 *20 = 400

فرمول عناصر غیر صفر : n+(n-1)+(n-1)=3n-2

عناصر غیر صفر :20 +19 + 19 = 58

عناصر صفر : 400 -58 = 342

نکته: ماتریس سه قطری گاهی خلوت است و گاهی خلوت نیست .

N*N	تعداد کل	غیر صفز	صفر
3*3	9	7	2
4*4	16	10	6
5*5	25	13	2
6*6	36	16	20

صف Queue

ساختمان داده ایست که عمل درج از یک طرف و عمل حذف از طرف دیگر صورت می گیرد .

قوانین

First in first out (FIFO)

Last in last out (LILO)

First come first service(FCFS)

برای نوشتن صف سه متغیر می خواهیم که یکی از آنهه بگوید چند خانه داریم ، یکی دیگر (front) که بگوید کدام عنصر اول است و دیگری (rear) که اشاره کند به عنصر آخری...

a	B	C	D	

front rear

اگر یک صف خالی باشد پس یعنی front و rear آن برابر 1- است.

نکته : در کنکور آرایه ها از یک شروع می شود پست front=rear=o

برای درج عنصر front و rear یکی به سمت جلو می روند .

Front ++;

Rear ++;

Queue[rear]= عنصر جدید;

برای درج عناصر بعدی دیگر front حرکت نمی کند فقط rear افزایش پیدا می کند .

شرط پر بودن صف

Rear = n;

حذف عنصر

برای این کار عنصر درون front حذف می شود و و عنصر بعدی جای آن را کمی گیرد مانند نانوایی.

A	B	C	D	

چه زمان صف یک عنصر دارد ؟

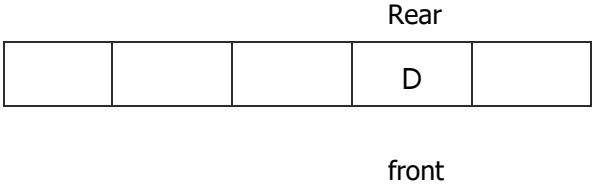
زمانی که :

Front = Reaer =1=-1

حذف آخرین عنصر باقی مانده

```
cout<< queue [front];
front=-1;
rear=-1;
```

نکته: زمانی که front=rear=n-1 یعنی صف علاوه بر این که تک عنصری است ، پر هم است و خانه های خالی قبل را عیب صف ساده می گویند .



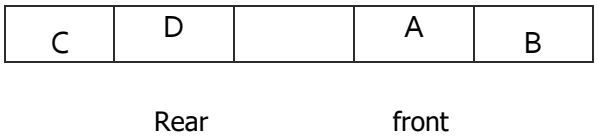
صف حلقوی

صفی است که امکان دور زدن صفر برای front و rear وجود دارد .

```
rear=(rear+1)%n;
queue [rear]=عنصر جدید;
```

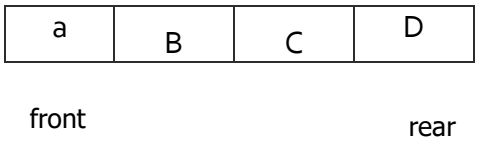
حذف از صف حلقوی

```
cout << queue[front];
front =(front+1)%n;
```

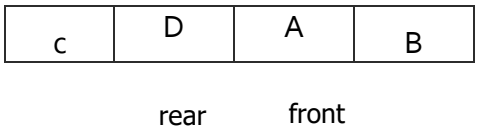


نکته : با این روش مشکل عیب صف ساده حل می شود .

شرط پر بود صف حلقوی



صف دور نزده= front ==0 && rear=-n-1;



صف دور زده= rear+1==front;

کاربرد های صف :

- درخواست های چاپگر
- در فایل های بسته ای batch

زبان ماشین : exe. و com.

دستورات سیستم عامل (غیراجرایی) : bat.

- در پیمایش سطح گراف (BFS)

پشته stack

ساختمان داده ایست که عمل درج و حذف از یک طرف صورت می گیرد .

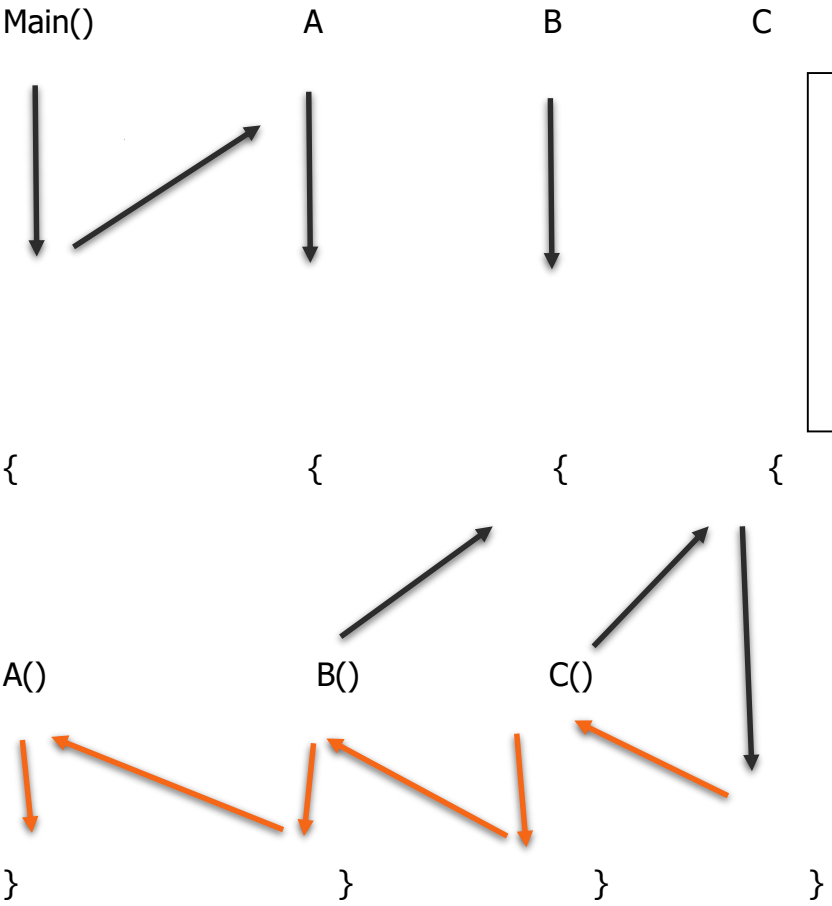
قانون

First in last out (FILO)

Last in first out (LIFO)

کاربرد های پشته

در فراخوانی توابع :



قبل از رفتن به A مسیر برگشت به MAIN را ذخیره کن

قبل از رفتن به B مسیر برگشت به A را ذخیره کن

قبل از رفتن به C مسیر برگشت به B را ذخیره کن

در توابع بازگشتی :

توابعی هستند که در بدنه خود ، خودرا فراخوانی می کنند.

```
A()
{
A()
}
```

غیر مستقیم

```
A()          B()
{            {
```

بطور مثال 5!=5*4*3*2*1 است و ما برای راحتی 5!6 را 5!6* می نویسیم .

```
B()                    A()
}                      }
```

مثال بازگشتی مستقیم :

```
fact (int n)
{
int i, f =1;
for (int I =1 ; i<=n ; i++)
{
f=f*i;
return f;
}
}
```

مثال از بازگشتی غیر مستقیم :

```
fact(int n)
{
if (n=-1)
return 1;
else
return n*fact(n-1);
}
```

مثال از بازگشتی غیر مستقیم :

```
fib (int n)
{
if (n<=1)
```

```
return n;
else
return fib (n-1)+fib (n-2);
}
```

پایان جلسه سوم

ساختمان داده

جلسه چهارم

ادامه کاربرد های پشته

در الگوریتم Maze :

الگوریتم Maze الگوریتمی :

بازگشتی درجه 4 است .

از پشته استفاده می کند .

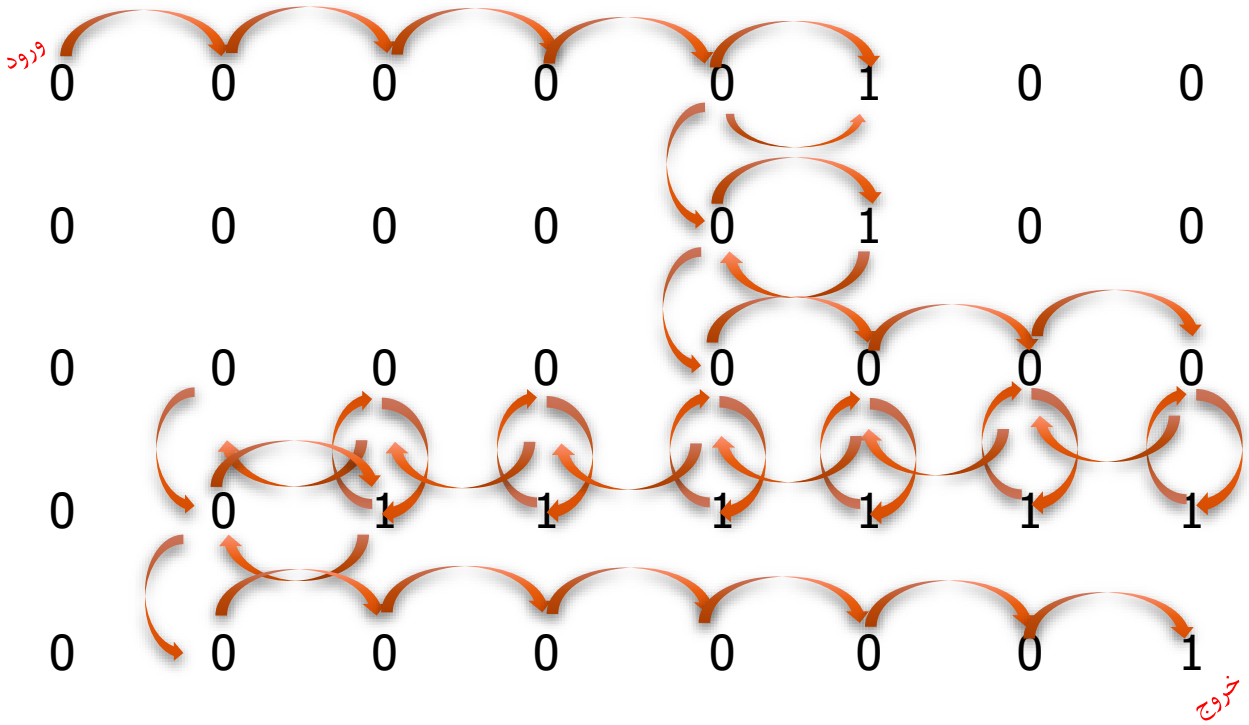
کند و کودم است.

حافظه بر است .

ول خوبی این الگوریتم این است که طول الگوریتم کم است .

0 یعنی مسیر باز است

و 1 یعنی مسیر بسته است



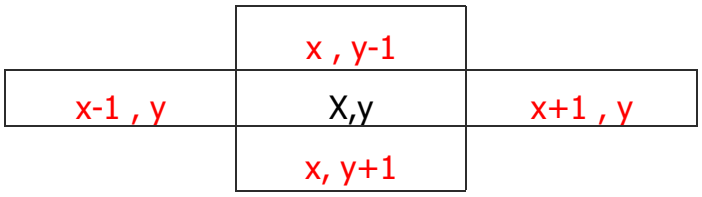
```
maze(int x,int y)

{

if (x==m-1 && y==n-1)

cout<<"Exit found";

if (map [y] [x] ==0 && x>=0 && x<m && y>=0 && y<n)

maze(x+1 ,y);

maze(x , y+1);

maze(x-1 , y);

maze(x , y-1);

}
```

نکته : در Maze اولویت با جلو است .

نحوه حرکت Maze :

1. روش 4 همسایگی:

در این روش الگوریتم 90 درجه جهت عوض می کند.



2. روش هشت همسایگی:

در این روش الگوریتم 45 درجه جهت عوض می کند .

	*	

نکته : در صفحه نمایش برای رفتن به بالای تصویر y-1 می شود و برای رفتن به پایین تصویر y+1 می شود .

در تبدیل عبارات محاسباتی

عبارات محاسباتی :

عملوند عملگر عملوند	میانوندی	Infix	A+B	استفاده در ریاضی
علوند عملگر عملوند	پیشوندی	Perifix	+AB	استفاده در کامپایلر
عملگر عملوند عملوند	پسوندی	Postfix	AB+	استفاده در کامپایلر

دلیل استفاده کامپایلر از عبارات بالا :

1.اولویت عملگر ها یکسان است.

2.فاقد پرانتز است .

در الگوریتم مرتب سازی سریع (Quick sort)

در الگوریتم مرتب سازی ادغامی (Mergesort)

در پیمایش عمقی گراف (DFS)

تیتر های بالا در جلسات بعدی توضیح داده می شوند .

پایان جلسه چهارم

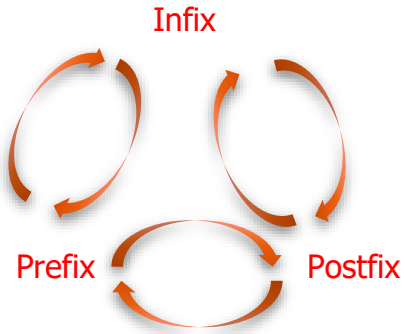
ساختمان داده

جلسه پنجم

تبدیل عبارات میانوندی به پیشوندی

برای تبدیل عبارات میانوندی به پیشوندی :

1. پرانتز گذاری کامل عبارت براساس اولویت عملگر ها
2. بردن هر عملگری پشت پرانتز باز خود
3. حذف پرانتز .



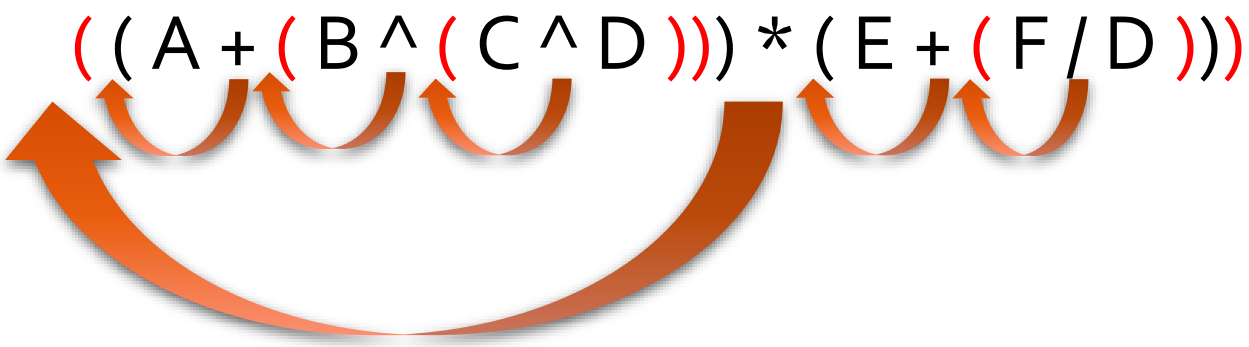
مثال : A + B * C –D?

جواب: ((A +(B * C)) –D)

خروجی: -+A*BCD

تست کنکور 82: (A +B ^ C ^ D) * (E + F / D)

جواب:



نکته: اگر در عبارات توان داشته باشیم باید از راست به چپ پرانتز گذاری کنیم ولی اگر توان نداشتیم از چپ به راست .

تبدیل عبارات پیشوندی به میانوندی

عملوند عملوند عملگر , +AB , پیشوندی , Prefix

تبدیل به

عملوند عملگر عملوند , A+B , میانوندی , Infix از سمت چپ عبارات حرکت می کنیم و ترتیب عملوند ها را اینگونه میکنیم

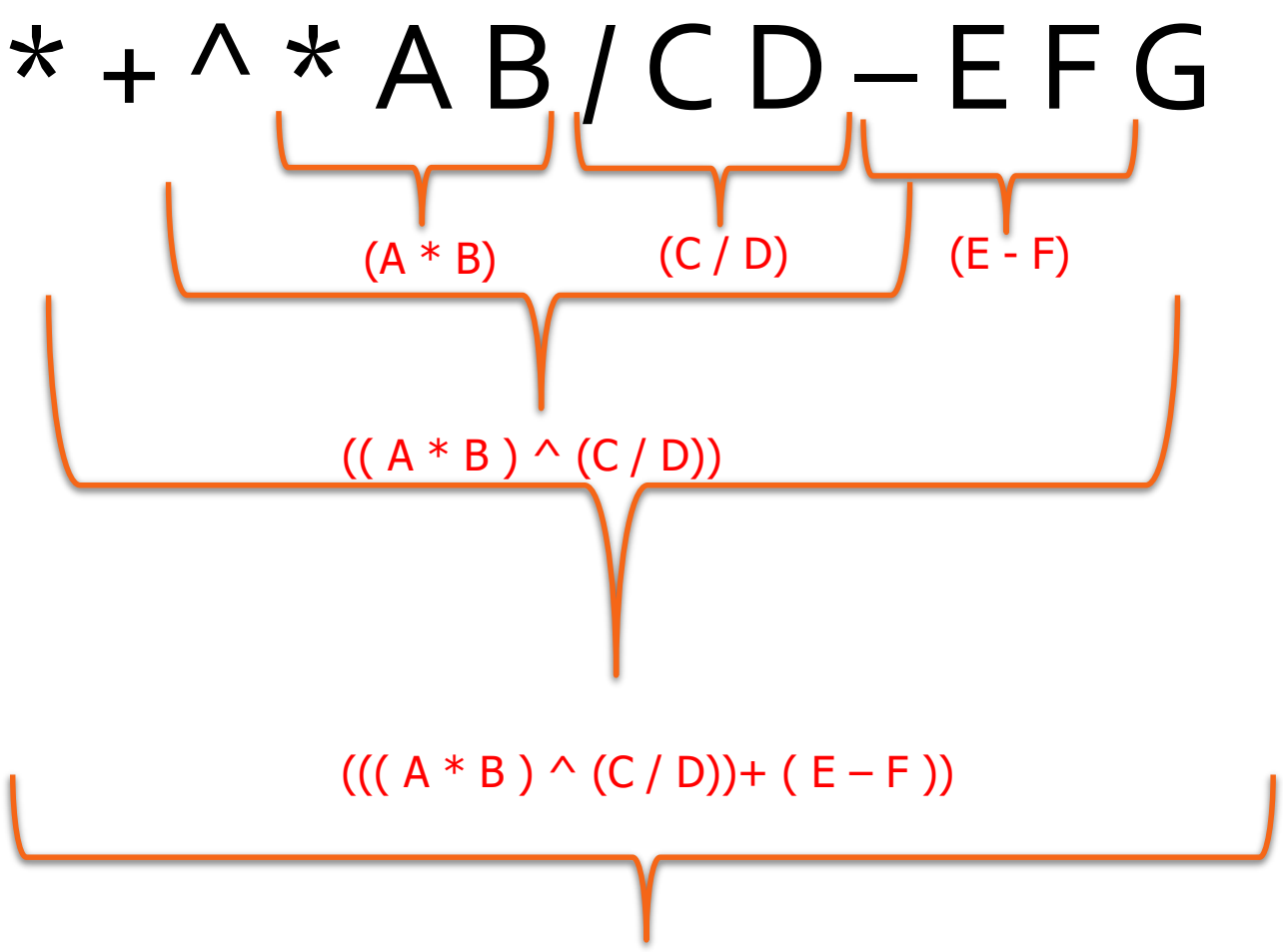
مثال :

*+ABC

جواب :

((A +B)* C)

سوال :



جواب پایانی :

(((A * B) ^ (C / D))+ (E – F)) * G)

تبدیل عبارات میانوندی به پسوندی

مراحل تبدیل عبارت میانوندی به پسوندی :

1. پرانتز گذاری کامل بر اساس اولویت عملگرها
2. بردن هر عملگری بعد از پرانتز بسته خود
3. حذف پرانتز ها

مثال :

A + B * C –D

((A +(B * C)) –D)

حاصل :

A B C * + D -

تست :

A + B * C * D / E - F + G ^ H

(((A + (((B * C) * D) / E)) - F) + (G ^ H))

جواب :

ABC * D * E / + F – G H ^ +

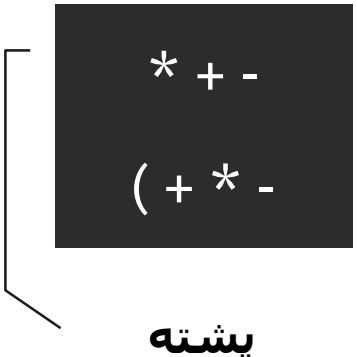
تبدیل عبارات میانوندی به پسوندی به کمک پشته

تستی : **A + B * C – D**

خروجی : **A B C * + D -**

دانشگاهی : **A + B * C - (D + E + F)**

خروجی: **ABC * DEF * + -**



چند نکته :

1. پرانتز باز تحت هر شرایطی وارد پشته می شود .
2. پرانتز باز پایین ترین اولویت را دارد .

الگوریتم تبدیل عبارت Infix به Postfix به کمک پشته

1. اگر عملوند در ورودی ببینیم در خروجی قرار می دهیم .
2. اگر پرانتز باز ببینیم در پشته قرار می دهیم .
3. اگر به عملگر برسیم و بالای پشته پرانتز باشد عملگر در پشته قرار می گیرد.
4. اگر به عملگر برسیم و اولویت آن از عملگر بالایی پشته بیشتر باشد آنرا در پشته قرار می دهیم . در غیر این صورت عملگر بالایی پشته را خارج کرده در خروجی قرار می دهیم و دوبار عمل بررسی را انجام می دهیم .
5. اگر به پرانتز بسته برسیم از بالای پشته خارج کرده در خروجی قرار می دهیم تا پرانتز باز برسیم . پرانتز باز را از پشته خارج کرده اما در خروجی قرار نمی دهیم .
6. اگر به انتهای عبارت برسیم محتوای پشته را در خروجی قرار می دهیم .

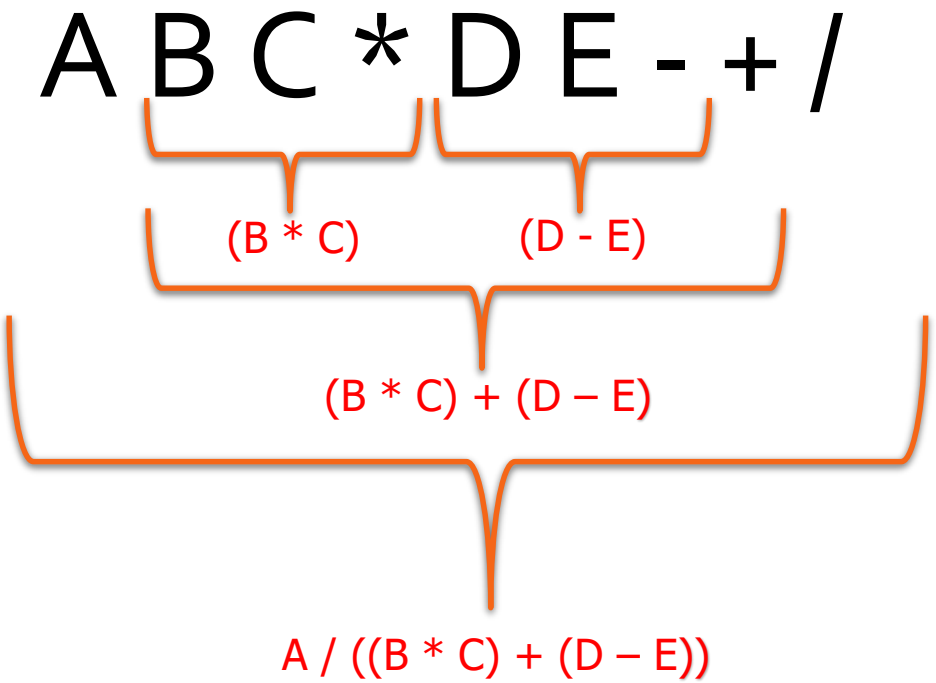
تبدیل عبارات پسوندی به میانوندی

عملوند ,عملوند ,عملگر AB+ , پسوندی

تبدیل به

عملوند ، عملگر ، عملوند A+B , میانوندی

مثال :



الگوریتم تبدیل عبارت پسوندی به میانوندی به روش سریع

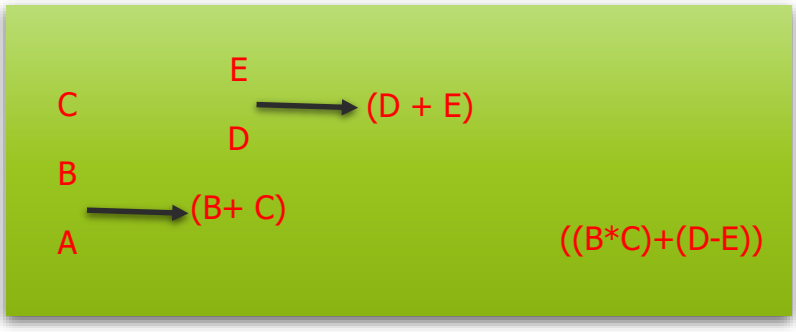
هرگاه تبدیل عملوند عملوند عملگر ببینیم آنرا تبدیل به عملوند عملگر عملوند می کنیم .

روش پشته

الگوریتم تبدیل عبارت پسوندی به کمک پشته

در این روش اگر به عملوند برسیم در پشته قرار می گیرد ، اگر به عملگر برسیم دو عملوند بالایی پشته را خارج کرده ، عملگر را مابین آن دو قرار می دهیم به همراه یک جفن پرانتز در پشته قرار می دهیم ، این کار تا آخر ورودی تکرار می شود و قتی به انتهای ورودی برسیم محتوای پشته را در خروجی قرار می دهیم .

ABC * DE - + /



پشته

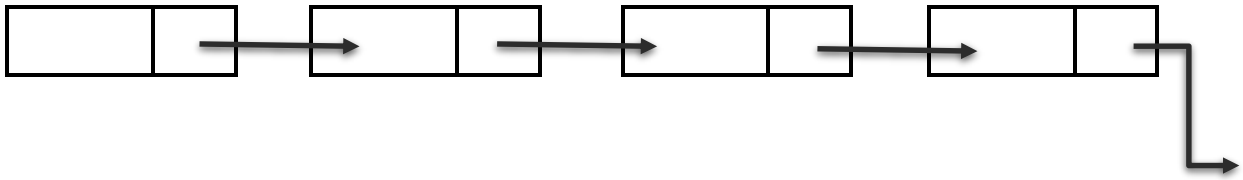
پایان جلسه پنجم

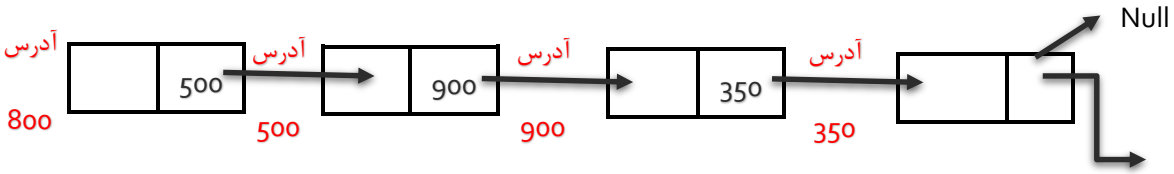
ساختمان داده

جلسه ششم

لیست پیوندی link list

ساختمان داده ایست پویا که متشکل از تعدادی گره است .

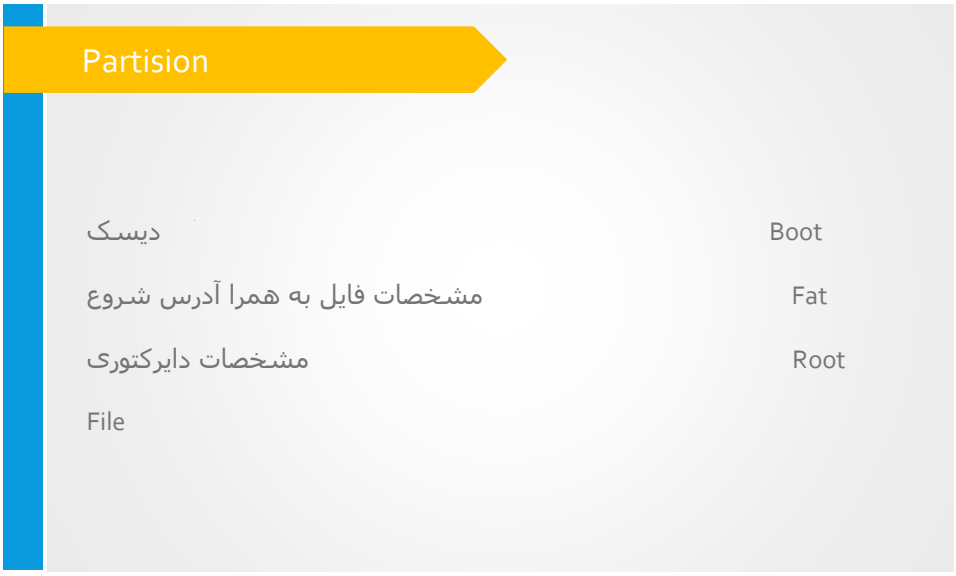




در این روش هر گره باید آدرس گره بعدی خود را بداند مانند صف نانوایی که باید نفر جلویی خود را بشناسیم .در سیستم عامل در مدیریت فایل ها روی دیسک نیز از این روش استفاده میشود .برای مثال زمانی که Defragment میکنیم فایل ها نامنظم میباشد برای منظم کردن آن فایل هر قسمت باید قسمت بعدی خود را بشناسد . در دیسک ها نحوه نشست فایل ها ناپیوسته است.

تعریف گره در C و C++

```
Struct نام { فیلدها

        } متغیر ها ;

Struct tnode { int data;

          tnode * next;

        } * start, *p,*q;
```



نکته : در لیست پیوندی دانستن آدرس گره اول بسیار مهم است .

نحوه تعریف اشاره گر در زبان C

نوع * نام ;

Int * P ;

نکته : در متغیر اشاره گر آدرس قرار می گیرد .

نکته : اگر start= null باشد یعنی لیست پیوندی گم شده است .

نکته : اگر P = start باشد یعنی به همتن جایی که start اشاره می کند ، P هم اشاره می کند .

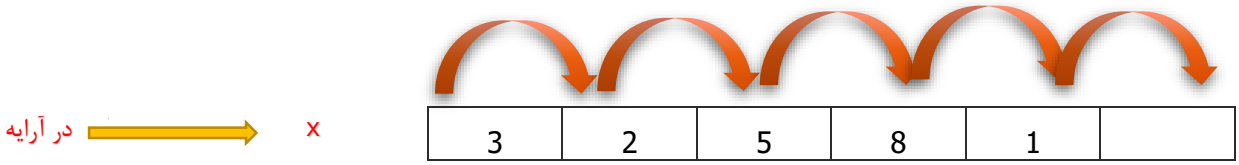
کارهای ویروس

1. تکثیر
2. تخریب

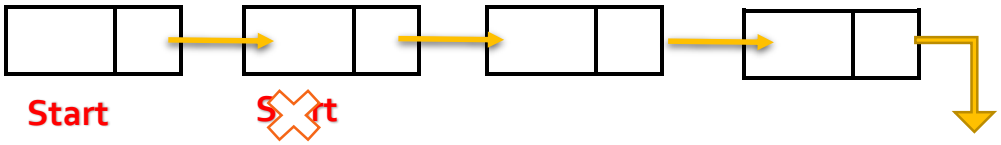
عملیات روی لیست پیوندی

1. درج
2. حذف

درج



در لیست پیوندی :



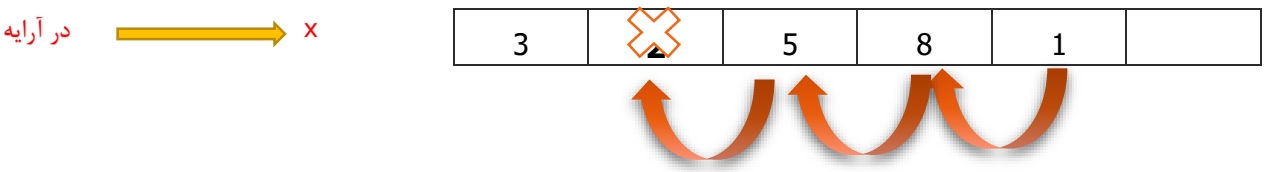
مزایای آرایه :

1. سرعت دستیابی بالا است.
2. کار با ان راحت تر است .

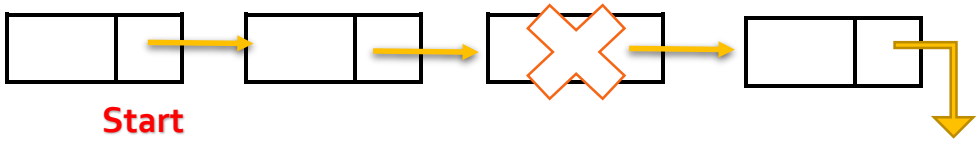
معایب آرایه :

1. ایستا است .
2. درج و حذف نیاز به شیفت دارد .

حذف



در لیست پیوندی :



مزایای لیست پیوندی :

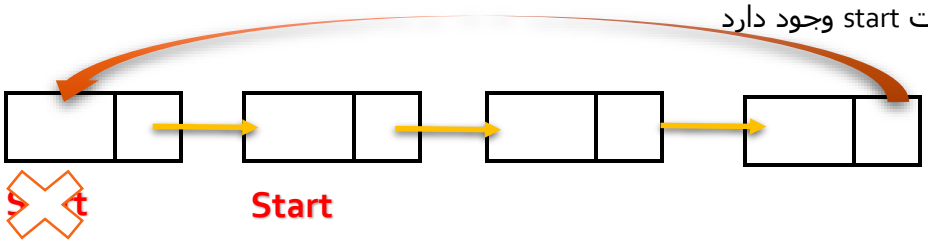
1. درج و حذف به راحتی صورت می گیرد .

معایب :

1. مصرف زیاد حافظه
2. مدیریت پیچیده اشاره گر ها

لیست پیوندی حلقوی

در این لیست امکان جرکت start وجود دارد

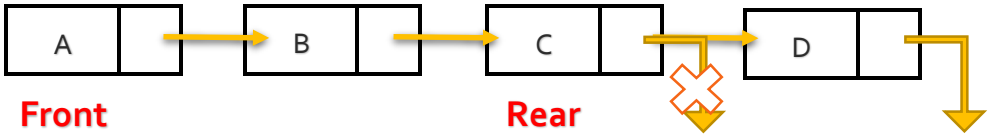


لیست پیوندی دوطرفه

مزیت لیست پیوندی دوطرفه :

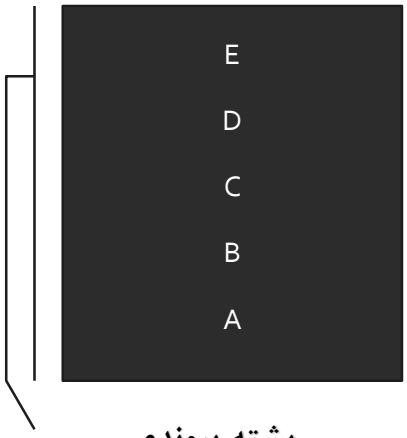
1. امکان بازگشت در لیست پیوندی داده می شود .

صف پیوندی : صفی است پویا که امکان درج از یک طرف(انتهای لیست)و امکان حذف از طرف دیگر(ابتدای لیست)امکان پذیر است.

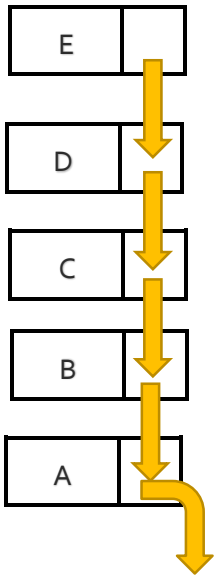


پشته پیوندی

ساختمان داده ای است پویا که امکان درج و حذف از یک طرف را می دهد.



پشته پیوندی



پایان جلسه ششم