فایل سوم درس ساختمان های داده

## دانشجویان عزیز کلاس درس ساختمان های داده

- با آرزوی سلامتی برای تمامی شما، در فایل ارسالی سعی گردیده تمامی مطالب، جهت مطالعه دقیق و انجام تمرین با جزییات بصورت تفصیلی و ارایه مثال مطرح گردد.

- امکان ارایه هرگونه سوال از صندوق پستی دانشگاه فراهم گردیده است.

موفق و سلامت باشید-اسبقی

## ساختمان های داده(Data Structures)

اصطلاحا به مفهوم سازماندهی منطقی داده ها در حافظه، ساختمان های داده گویند. قبلا در کلاس درخصوص اهمیت داده و نقش آن در دنیای امروز و رشته مهندسی کامپیوتر صحبت شده و برهمین اساس، درس ساختمان های داده نیز در این رشته اهمیت فراوانی دارد. برای روشن شدن بهتر مفهوم سازماندهی داده ها، نیاز به مباحثی مانند انواع داده ای وجود دارد.

### انواع داده ای(Data Types):

- انواع داده ای ساده

انواع داده ای ساده در زبان c++: int, float, double, bool, char

- انواع داده ای پیچیده یا تعریف شده توسط کابر(User Defined)

این انواع داده ای بصورت مجموعه ای بوده و توسط کاربران بسته به نیاز برنامه تعریف می شوند.

مثال: نوع داده ای مجموعه ای(همگن):Array, string

مثال: نوع داده ای مجموعه ای (ناهمگن):struct, class

بعنوان مثال برای نوع داده struct داریم:

```
struct   نام{
  بدنه ساختمان
};

struct   student{
  char    std_name[20];
  int     std_no;
  float   std_avg;};
```

برای ایجاد متغیر همواره بدین صورت عمل کرده ایم:

نوع داده ای  متغیر;

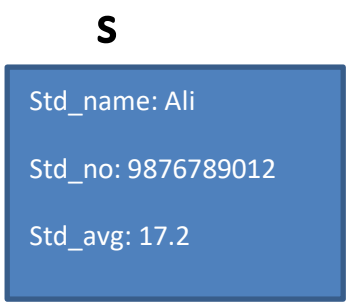
بعنوان مثال:

```
int   x;
```

این دستور متغیری را به سایز(2Bیا4B ) برای ذخیره یک عدد صحیح بر روی حافظه تعریف میکند.(در این دستور نمودی از جزییات حافظه و پیاده سازی قابل مشاهده است، ولی هنگامی که متغیری از نوع ساختمان دانشجو بصورت زیر تعریف می شود:

```
student    s;
s.std_name = “Ali”;
s.std_no = 9876789012;

s.std_avg = 17.2;
```




همانطور که مشاهده می شود، متغیر s بصورت یک متغیر مجموعه ای با سه بخش مختلف ایجاد شده است، بطوریکه در نگاه اول، همانند دنیای واقعی پیرامون ما، متغیری از نوع ساختمان دانشجو تعریف نموده که در قدم اول جزییات پیاده سازی را در نظر نگرفته است. بر همین اساس، مفهوم نوع داده انتزاعی بصورت زیر تعریف می گردد:

## مفهوم نوع داده انتزاعی(Abstract Data Type)(ADT):

هر نوع داده ای، مجموعه ای مجاز از مقادیر داده و مجموعه عملیات مجاز قابل اعمال بر روی آن داده ها را مشخص میکند، بعنوان مثال int x; مشخص می کند که متغیر x می تواند مجموعه ای از مقادیر عدد صحیح را پذیرا بوده و فقط مجموعه عملیات مجاز بر روی اعداد صحیح می تواند ازین پس بر روی x نیز اعمال گردد. یا مثلا در مورد نوع داده کلاسها نیز عنوان می شد، هر

کلاس مجموعه ای از داده ها و عملیات مجاز برروی آنهاست. حال هر گاه این انواع داده ای، بدون جزییات پیاده سازی و با توجه به مفاهیم دنیای پیرامون ما مطرح شوند، آنها را نوع داده انتزاعی گویند. کلاس "دانشجو" یا ساختمان "دانشجو" بسته به مفهوم واقعی دانشجو در محیط تعریف شده و بدور از جزییات سخت افزاری و پیاده سازی می باشد.

## کاربرد ADT در ساختمان های داده:

براساس تعریف بالا، در درس ساختمان های داده، برای یادگیری بهتر، هر مفهومی ابتدا بصورت انتزاعی و با مفاهیم مانوس در دنیای واقعی مطرح شده و سپس به جزییات پیاده سازی آن پرداخته می شود.

## آرایه(Array):

مفهوم آرایه را از درس برنامه نویسی به خاطر داریم:

```
int   A[5];
```

در این تعریف آرایه، یک مجموعه 5 تایی با نام A برای ذخیره اعداد صحیح بر روی حافظه RAM ایجاد می گردد و تمامی عملیات مجاز اعداد صحیح، می تواند بر روی داده های این آرایه نیز اعمال گردد. همانطور که مشاهده می کنید این تعریف انتزاعی نیست و تا حدی به جزییات سخت افزاری و حافظه می پردازد.

## آرایه بعنوان ساده ترین ساختمان داده:

## تعریف آرایه به عنوان یک ADT:

**مجموعه ای محدود و مرتب از عناصر همگن.** منظور از محدود بودن آرایه، سایز ثابت آرایه است که در ابتدای تعریف آن باید مشخص شود، و همینطور عناصر آرایه مرتب می باشد یعنی عناصر آرایه نسبت به هم ترتیب خاصی دارند و منظور از همگن بودن عناصر نیز، هم نوع بودن

آنهاست، یعنی همه عناصر آرایه باید بعنوان مثال عدد صحیح یا اعشاری باشند و نمی توان ترکیبی از آنها را در یک آرایه داشت.

## عملیات:

دسترسی مستقیم به عناصر آرایه

## انواع دسترسی:

- **دسترسی ترتیبی**

در دسترسی ترتیبی، برای رسیدن به هر عنصر، بایستی به ترتیب تمامی عناصر قبلی پیمایش شوند تا به عنصر مورد نظر بتوان دست یافت. بعنوان مثال نوار کاست، مجموعه ای از فایلهای صوتی را دارد که دسترسی در آن ترتیبی است، بطوریکه برای رسیدن بعنوان مثال به فایل صوتی پنجم، باید چهار فایل قبلی رد شود تا به فایل پنجم بتوان دسترسی داشت. در این روش، بدیهی است هزینه دستیابی به عنصر اول با صدم متفاوت خواهد بود.

- **دسترسی مستقیم/تصادفی**

در این روش، دسترسی  مستقیما به عنصر مورد نظر صورت می گیرد، بطوریکه هزینه دستیابی به عنصر اول با عنصر صدم تفاوتی ندارد.

حال باید دید این روش دسترسی مستقیم چگونه پیاده سازی می شود و نحوه آدرس دهی در آن به چه صورتی است؟

## نحوه آدرس دهی در آرایه ها و دسترسی مستقیم:

### آرایه خطی:

آرایه خطی، همان آرایه تک بعدی است که بصورت روبرو تعریف می شود:

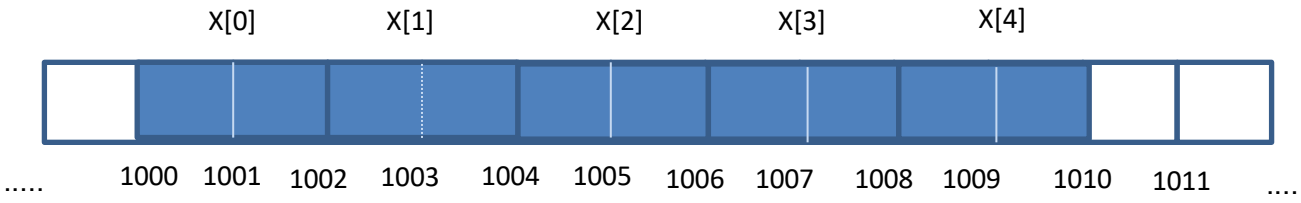
```
int  x[5];
```




### پیاده سازی آرایه در حافظه:

- **ذخیره سازی به روش خطی**

در این روش آرایه 5 تایی X بصورت زیر در حافظه ذخیره می گردد:

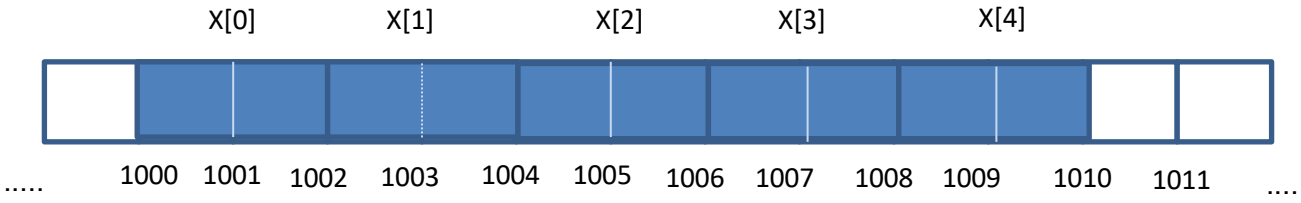



شکل بالا بعنوان مثال بخشی از حافظه را نمایش می دهد، همانطور که مشخص است تقسیم بندی حافظه براساس بایت بوده و هر بایت نیز دارای یک آدرس می باشد. آرایه x بصورت خطی با شروع از آدرس 1000 فرضی و در نظر گرفتن سایز int به اندازه 2 بایت ذخیره شده است.

### نحوه آدرس دهی:

**آدرس x[i] = base + { (i-l) * size}**

منظور از base، آدرس شروع آرایه است که در مثال بالا 1000 درنظر گرفته شده است و منظور از size، تعداد بایت درنظر گرفته شده برای هر عدد صحیح می باشد. l نیز به اندیس شروع آرایه اشاره دارد که در زبان c++، اندیس شروع آرایه از 0 می باشد پس نیازی به نوشتن آن در ادامه نمی باشد.




**آدرس x[i] = base + { (i-l) * size}**

بعنوان مثال داریم:

آدرس x[2] = 1000 + {(2 -0) * 2} ⟹ 1004

آدرس x[4] = 1000 + {(4 -0) * 2} ⟹ 1008

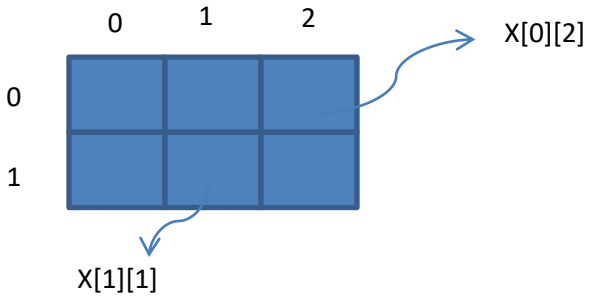
آدرسهای بدست آمده، آدرس شروع عناصر مذکور آرایه است، در این روش می توان دید که دسترسی مستقیم بوده و تفاوتی در دستیابی به عناصر مختلف وجود ندارد.

## آرایه دو بعدی(ماتریس):

آرایه دو بعدی بصورت زیر تعریف می شود:

```
int  x[2][3];

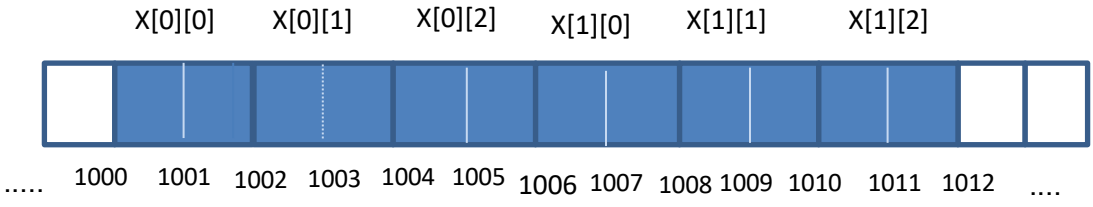
int x[d1][d2];
```




بعد d1 به بعد اول آرایه دوبعدی یعنی تعداد سطر و بعد دوم d2 به تعداد ستون های ماتریس اشاره دارد. در ماتریس ها از آنجاییکه یک فضای دوبعدی را تصویر می کنند، برای ذخیره در فضای یک بعدی حافظه، نیاز به نگاشت به فضای تک بعدی را دارند که بر این اساس دو روش ذخیره سطری و ستونی بصورت زیر وجود دارد:

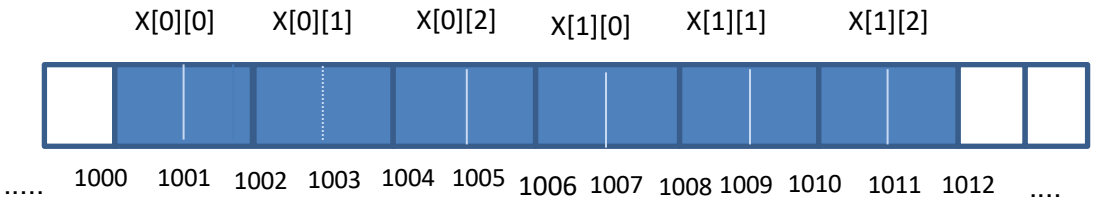
- **ذخیره سازی به روش سطری**

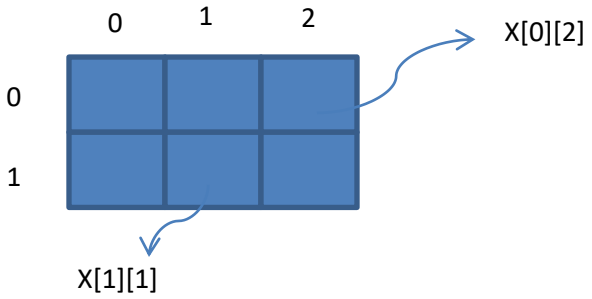
در این روش ماتریس x به روش افقی و سطری در حافظه ذخیره می گردد:




شکل بالا بعنوان مثال بخشی از حافظه را نمایش می دهد که ماتریس X بصورت سطر به سطر و پشت سر هم در حافظه ذخیره شده است. همانطور که مشخص است، آرایه دوبعدی X بصورت

خطی با شروع از آدرس 1000 فرضی و در نظر گرفتن سایز int به اندازه 2 بایت ذخیره شده است.




## نحوه آدرس دهی در روش سطری:

**آدرس x[i][j] = base +{ { (i * d2) + j} * size}**

بعنوان مثال داریم:

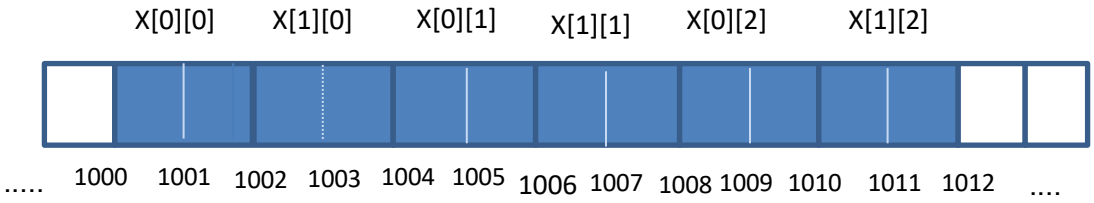
آدرس x[0][2] = 1000 + {{(0*3) + 2}* 2} ⟹ 1004

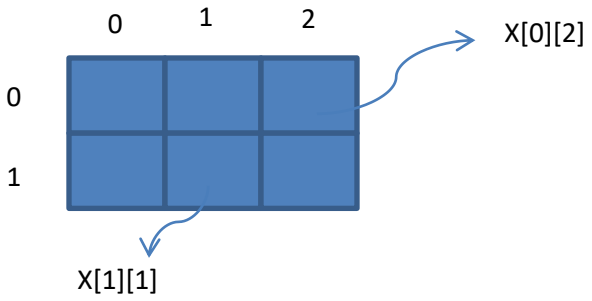
آدرس x[1][1] = 1000 + {{(1*3) + 1}* 2} ⟹ 1008

آدرسهای بدست آمده، آدرس شروع عناصر مذکور آرایه است و با توجه به شکل بالا، صحت آنها قابل اثبات است.

- **ذخیره سازی به روش ستونی:**

در این روش ماتریس x به روش عمودی و ستونی در حافظه ذخیره می گردد:




مجددا شکل بالا بعنوان مثال بخشی از حافظه را نمایش می دهد که ماتریس x بصورت ستون به ستون و پشت سر هم در حافظه ذخیره شده است. یعنی ابتدا ستون اول، سپس ستون دوم پست سر آن و به همین ترتیب... همانطور که مشخص است، آرایه دوبعدی x بصورت ستونی با شروع از آدرس 1000 فرضی و در نظر گرفتن سایز int به اندازه 2 بایت ذخیره شده است. به آدرسها توجه شود.

**نحوه آدرس دهی در روش ستونی:**

**آدرس x[i][j] = base +{ { (j * d1) + i} * size}**

بعنوان مثال داریم:

آدرس x[0][2] = 1000 + {{(2*2) + 0}* 2} ⟹ 1008

آدرس x[1][1] = 1000 + {{(1*2) + 1}* 2} ⟹ 1006

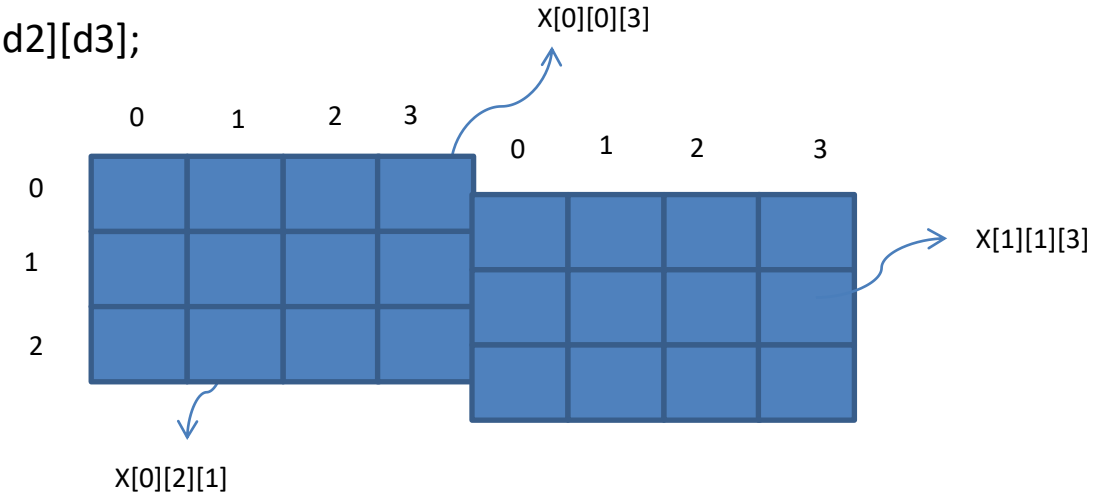
آدرسهای بدست آمده، آدرس شروع عناصر مذکور آرایه است و با توجه به شکل بالا، صحت آنها نیز قابل اثبات است.

## آرایه سه بعدی:

در تعریف یک آرایه سه بعدی، بعد اول d1، تعداد صفحات، بعد دوم d2، تعداد سطرها و بعد سوم d3، تعداد ستونها را نشان میدهد. بعنوان مثال داریم:

```
int  x[2][3][4];

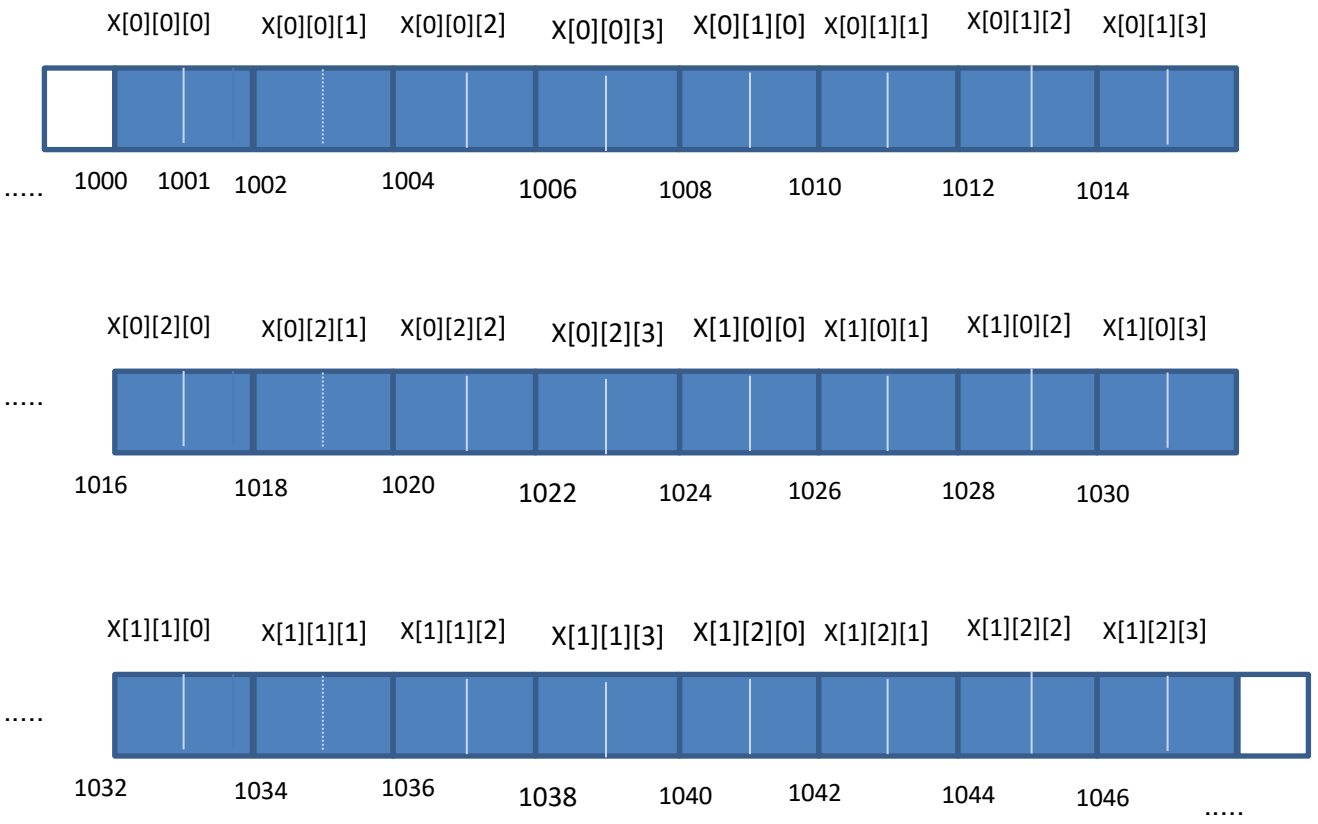
int   x[d1][d2][d3];
```

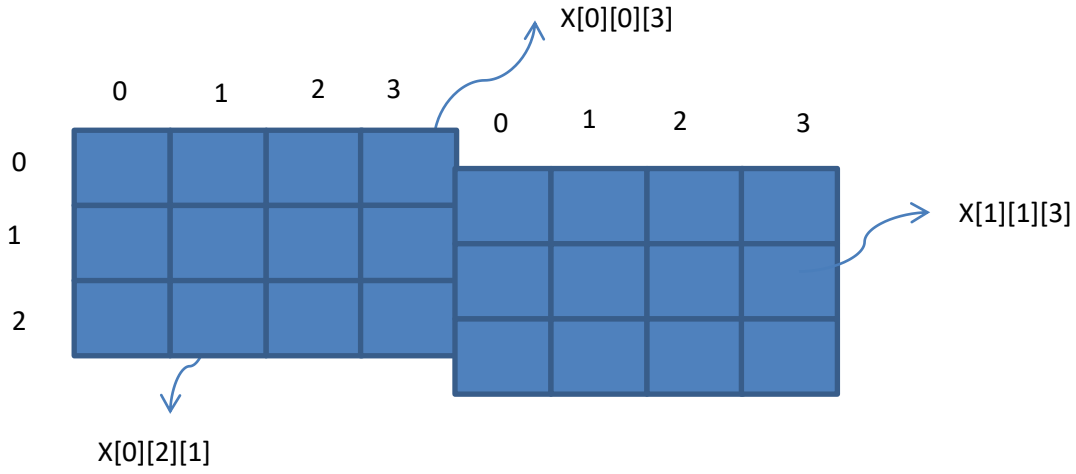



در آرایه های سه بعدی نیز برای ذخیره سازی در حافظه یک بعدی، نیاز به نگاشت به فضای تک بعدی را دارند که بر این اساس دو روش ذخیره سطری و ستونی بصورت زیر وجود دارد:

- **ذخیره سازی به روش سطری**

در این روش آرایه سه بعدی X به روش افقی و سطری در حافظه ذخیره می گردد، بدین معنی که ابتدا صفحه اول بصورت سطر به سطر ذخیره شده و سپس صفحه دوم و سطرهایش پشت سر آنها ذخیره می گردند:

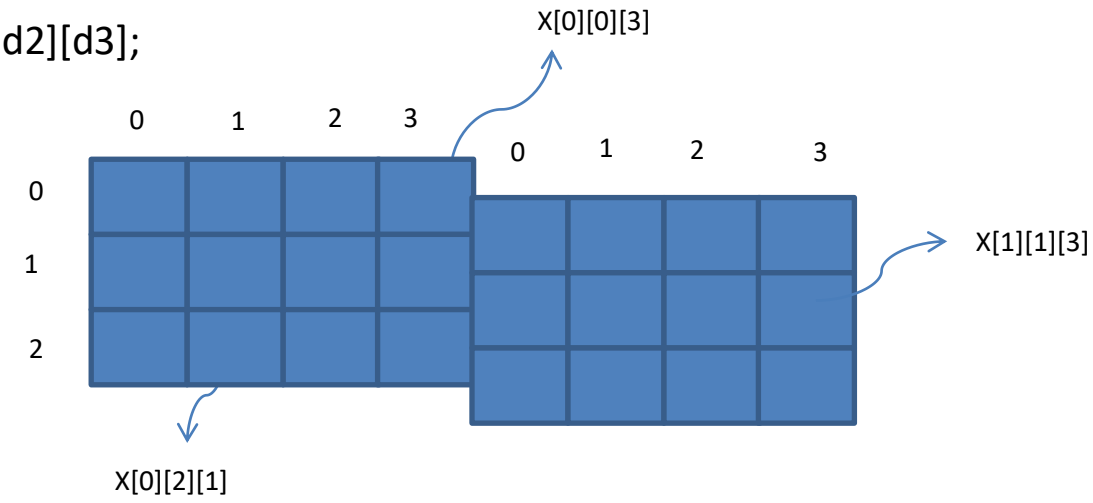


شکل بالا بعنوان مثال بخشی از حافظه را نمایش می دهد که آرایه سه بعدی X بصورت سطر به سطر و پشت سر هم در حافظه ذخیره شده است.

```
int  x[2][3][4];

int   x[d1][d2][d3];
```




## نحوه آدرس دهی در روش سطری:

**آدرس x[i][j][k] = base +{ { (i * d2 * d3) + (j *d3) + k } * size}**

بعنوان مثال داریم:

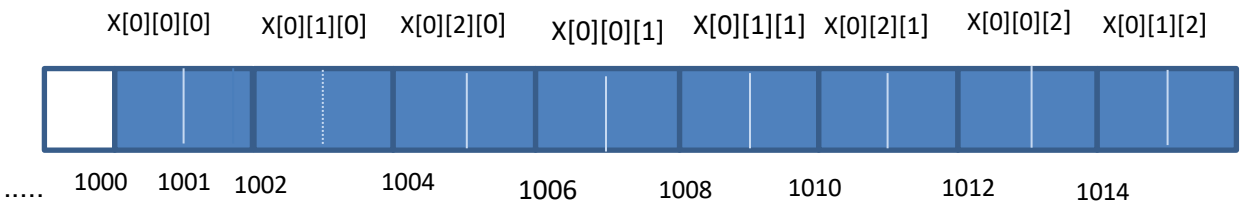
آدرس x[0][2][1] = 1000 + {{(0*3*4) + (2*4) + 1}* 2} ⟶ 1018

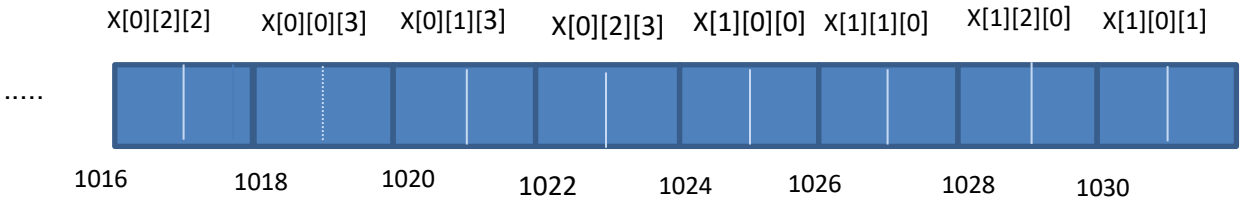
آدرس x[1][1][3] = 1000 + {{(1*3*4) + (1*4) + 3}* 2} ⟶ 1038

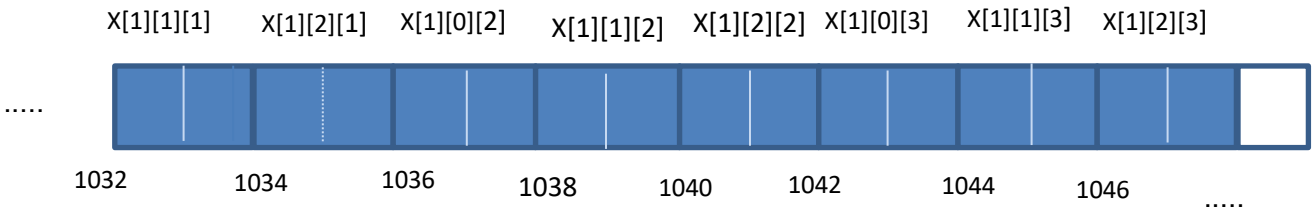
آدرسهای بدست آمده، آدرس شروع عناصر مذکور آرایه است و با توجه به شکل بالا، صحت آنها نیز قابل اثبات است.

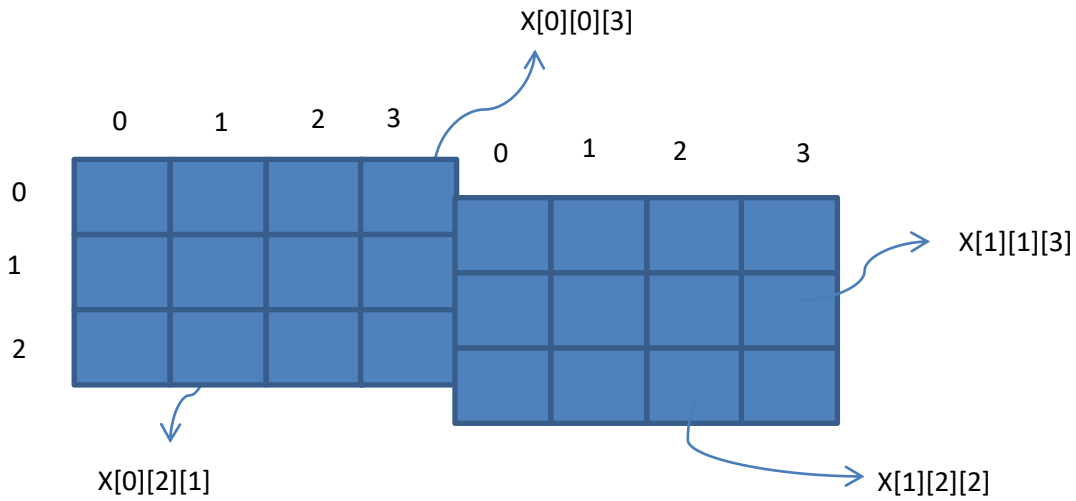
- **ذخیره سازی به روش ستونی:**

در این روش ماتریس **X** به روش عمودی و ستونی در حافظه ذخیره می گردد:



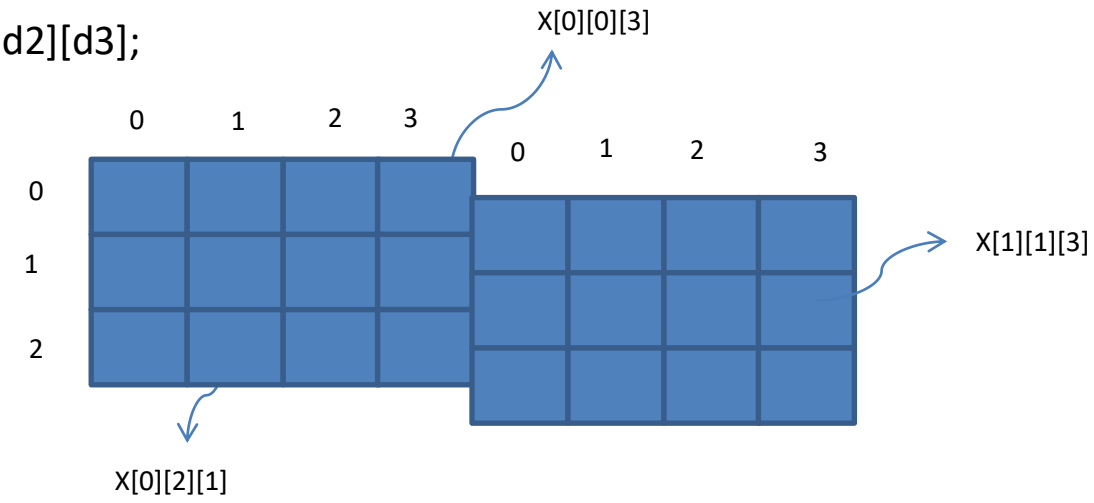



شکل بالا بعنوان مثال بخشی از حافظه را نمایش می دهد که آرایه سه بعدی X بصورت ستون به ستون و پشت سر هم در حافظه ذخیره شده است.

```
int  x[2][3][4];

int   x[d1][d2][d3];
```




## نحوه آدرس دهی در روش ستونی:

**آدرس x[i][j][k] = base +{ { (i * d2 * d3) + (k *d2) + j } * size}**

بعنوان مثال داریم:

آدرس x[0][2][1] = 1000 + {{(0*3*4) + (1*3) + 2}* 2} ⟶ 1010

آدرس x[1][1][3] = 1000 + {{(1*3*4) + (3*3) + 1}* 2} ⟶ 1044

## ماتریس اسپارس(خلوت):

ماتریس زیر را در نظر بگیرید:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 9 & 0 & 0 \end{pmatrix}$$

تعداد عناصر مفید در ماتریس بالا به نسبت تعداد عناصر غیر مفید بسیار پایین می باشد.

**میزان حافظه مصرفی:** 5 * 6 * 2 byte

**میزان حافظه مورد نیاز برای عناصر مفید:** 4 * 2 byte

نحوه نمایش ماتریس اسپارس:

| تعداد عناصر غیر صفر | تعداد ستون | تعداد سطر |
|---|---|---|
| مقدار عنصر غیر صفر | ستون | سطر |

| 4 | 6 | 5 |
|---|---|---|
| 5 | 2 | 1 |
| 7 | 1 | 2 |
| 4 | 5 | 3 |
| 9 | 3 | 4 |

**میزان حافظه مصرفی ماتریس اسپارس:**

5 * 3 * 2 byte

سوال: اگر آرایه 4 بعدی A را بصورت زیر درنظر بگیریم، آدرس A[1][1][2][1] را در شرایطی بدست آورید که آرایه به روش سطری در حافظه ذخیره شده باشد. آدرس شروع آرایه را 1000 و سایز int را 2 بایت در نظر بگیرید.

```
int  A[2][3][2][4];
```

سوال: بررسی نمایید که ماتریس قطری، یک ماتریس اسپارس است؟

ماتریس قطری

$$A = \begin{pmatrix} a00 & 0 & 0 \\ 0 & a11 & 0 \\ 0 & 0 & a22 \end{pmatrix}$$