آرایه ها و ساختارها

آرایه

❖آرایه مجموعه ای از زوج ها ، شامل اندیس و مقدار است (<index , value>). به ازای هر اندیس یک مقدار مربوط به آن اندیس وجود دارد که به زبان ریاضی تناظر یا نگاشت نامیده می شود.

❖ **در رابطه با آرایه به دو عمل اساسی نیاز است**

- بازیابی
- ذخیره سازی مقادیر

آرایه

❖ **تابع Create:** یک آرایه جدید تهی با طول مناسب را تولید می کند. تمام مقادیر در ابتدا تعریف نشده اند.

❖ **تابع Retrieve:** یک آرایه و یک اندیس را به عنوان ورودی دریافت می کند و یک مقدار مربوط به اندیس را اگر اندیس معتبر باشد برمیگرداند و گرنه یک خطا را بازمی گرداند.

❖ **تابع store:** برای وارد کردن زوج جدیدی شامل < اندیس، مقدار> به کار می رود و آرایه اولیه افزایش یافته با زوج جدید،<مقدار،اندیس> را بازمی گرداند.

آرایه

آرایه در زبان C به صورت مثال در زیر آمده است :

```
int list [5];
```

نکته

در زبان C تمام آرایه ها از اندیس ۰ شروع می شوند.

آدرس عناصر آرایه یک بعدی

- آرایه x به طول n داریم
- آدرس عنصر اول = α
- اندازه هر عنصر = β
- آدرس عنصر x[i]=?

ساختارها

- آرایه ها مجموع داده های از یک نوع می باشند. در C روش دیگری برای دسته بندی گروهی از داده وجود دارد . این روش اجازه می دهد تا داده ها از انواع متفاوتی باشند. این امکان را struct می گویند که مختصر structure است.

- یک ساختار مجموعه ای از اقلام داده ها می باشد که هر قلم داده به وسیله نوع و نام آن مشخص گردیده است.

ساختارها

مثال:

متغیری به نام person ایجاد می کند که دارای سه فیلد متفاوت است :

- نام شخص به صورت یک آرایه کاراکتری است .
- یک مقدار صحیح که نشان دهنده سن همان شخص است.
- یک مقدار float که حقوق شخص را معین می کند.

```
struct{
        char name[10];
        int age;
        float salary;
        }person

strcpy(person.name, “james”);
person.age=10;
person.salary=35000;
```

ساختارها

```
typedef struct human_being {
char name[10];
int age;
float salary;
};
```

or

```
typedef struct {
char name[10];
int age;
float salary
} human_being;

human_being person1, person2;
```

ساختارهای خود ارجاعی

❖ یک ساختار خودارجاعی ساختاری است که در آن یک جز یا بیشتر ، اشاره گری به خود آن می باشد.

❖ ساختارهای خود ارجاعی معمولا به روالهای مدیریت حافظه پویا احتیاج دارند , (malloc free) تا به راحتی حافظه را گرفته و آزاد کنند.

```
typedef struct list {
        char data;
        list *link;};
```

یونیون ها

❖ اعلان یونیون مشابه تعریف و اعلان یک ساختار است با این تفاوت که فیلدهای یک یونیون باید در حافظه با هم مشترک باشند یعنی فقط یک فیلد یونیون در هر زمان فعال می گردد.

```
union u {
        int x;
        int y;
 }
```

لیست

❖ ساده ترین و متداول ترین نوع ساختمان داده ها ، لیست های مرتب شده یا خطی هستند.

❖لیست ها شامل اقلام داده به صورت $(item_0, item_1, ..., item_{n-1})$ می باشند.

❖ مثال :

روزهای هفته (شنبه ... جمعه)

اعمال صورت گرفته بر روی لیست ها

- پیدا کردن طول یک لیست
- خواندن اقلام داده یک لیست از چپ به راست یا بر عکس
- بازیابی i امین عنصر از یک لیست $(0 \leq i < n)$
- تعویض یک قلم اطلاعاتی در i امین موقعیت یک لیست $(0 \leq i \leq n)$
- درج یک قلم داده جدید در i امین موقعیت یک لیست $(0 \leq i < n)$.

(اقلام داده ای که قبلا به صورت i، i+1،...،n-1 شماره گذاری شده اند به صورت i+1،i+2،...،n در می آیند)

- حذف یک قلم اطلاعاتی از i امین موقعیت یک لیست $(0 \leq i < n)$. اقلام داده i+1،...،*n-1* به اقلام داده با شماره *I*،*i+1*،...،*n-2* تبدیل می شود.

نگاشت ترتیبی

❖ متداول ترین پیاده سازی ، نمایش یک لیست مرتب شده به صورت یک آرایه می باشد به نحوی که عنصر $item_1$ لیست با اندیکس i آرایه متناظر باشد. این مطلب یک نگاشت ترتیبی نامیده می شود.

ماتریس اسپارس

❖ به طور کلی در ریاضیات ، یک ماتریس شامل m سطر و n ستون بوده و می تواند مانند شکل زیر نمایش داده شود.

	col0	*col1*	*col2*
row 0	-۲۷	۳	۴
row 1	۶	۸۲	-۲
row 2	۱۰۹	-۶۴	۱۱
row 3	۱۲	۸	۹
row 4	۴۸	۲۷	۴۷

ماتریس اسپارس

❖ در علوم کامپیوتر متداول ترین نمایش برای ماتریس آرایه دوبعدی است که به صورت a[MAX_ROW][MAX_COLS] نمایش داده می شود. هر عنصر ماتریس به صورت a[i][j] نمایش داده می شود. ماتریسی که عناصر صفر آن زیاد بوده ماتریس اسپارس نامیده می شود.

	col1	col2	col3	col4	col5	col6
row0	15	0	0	22	0	−15
row1	0	11	3	0	0	0
row2	0	0	0	−6	0	0
row3	0	0	0	0	0	0
row4	91	0	0	0	0	0
row5	0	0	28	0	0	0

❖ حداقل اعمال ممکن شامل **ایجاد، جمع ، ضرب و ترانهاده ماتریس** می باشد.

ذخیره ماتریس اسپارس

❖ می توان هر آرایه را با استفاده از سه گانه ذخیره نمود، این بدان معنا می باشد که می توان یک آرایه از سه گانه ها را برای نمایش یک ماتریس اسپارس پیشنهاد کرد.

```
#define MAX_TERMS 101 /*
maximum number of terms */
    typedef struct {
                int col;
                int row;
                int value;
                } term;
    term a[MAX_TERMS];
```

ذخیره ماتریس اسپارس

❖ می توان هر آرایه را با استفاده از سه گانه ذخیره نمود این بدان معنا می باشد که می توان یک آرایه از سه گانه ها را برای نمایش یک ماتریس اسپارس پیشنهاد کرد.

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

	row	col	value
	# of rows	# of columns	# of nonzero terms
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

ترانهاده ماتریس اسپارس

❖ برای پیدا نمودن ترانهاده یک ماتریس باید جای سطرها و ستون ها را عوض کرد بدین مفهوم که هر عنصر a[i][j] در ماتریس اولیه به عنصر b[j][i] در ماتریس ترانهاده تبدیل می شود.

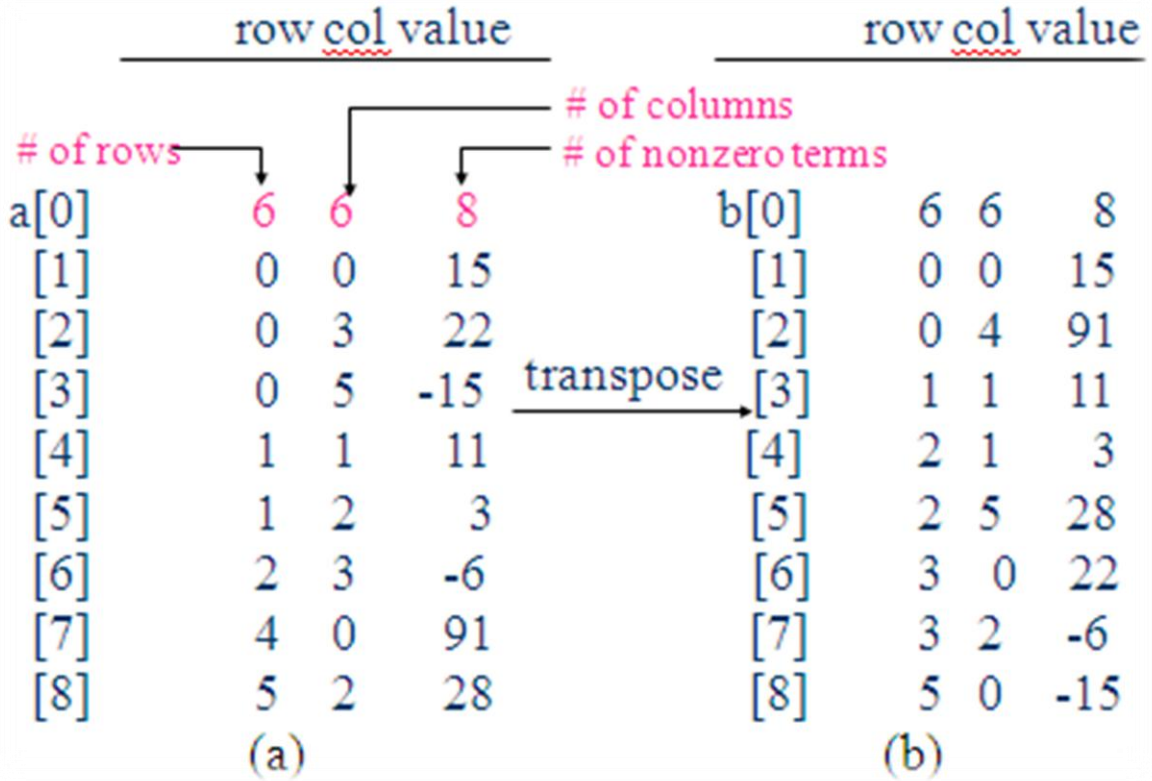
✓ الگوریتم زیر برای پیدا کردن ترانهاده یک ماتریس ، الگوریتم مناسبی است :

```
for all element is column j
        place  element < i  ،j ،value> in
        element < j  ،i ،value>
```

ترانهاده ماتریس اسپارس

❖ الگوریتم بیان شده نشان می دهد که باید تمام عناصر در ستون ۰ را پیدا و آنها را در سطر ۰ ذخیره کرد همچنین تمام عناصر ستون ۱ را پیدا و در سطر ۱ قرار داد و همین فرآیند را ادامه داد. از آنجا که ماتریس اولیه سطری بوده لذا ستون های داخل هر سطر از ماتریس ترانهاده نیز به صورت صعودی مرتب می شود.

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$



ترانهاده ماتریس اسپارس

```
void transpose (term a[], term b[])
{   int n, i, j, currentb;
    n = a[0].value;
    b[0].row = a[0].col;
    b[0].col = a[0].row;
    b[0].value = n;
    if (n > 0) {
        currentb = 1;
        for (i = 0; i < a[0].col; i++)
            for( j = 1; j <=  n; j++)
                if (a[j].col == i) {
                        b[currentb].row = a[j].col;
                        b[currentb].col  = a[j].row;
                        b[currentb].value = a[j].value;
                        currentb++
                }
    }
}
```

تحلیل ترانهاده ماتریس اسپارس

❖ تعیین زمان اجرای این الگوریتم از آنجا که حلقه های تودرتوی for عامل تعیین کننده می باشد ، آسان است.

❖حلقه for خارجی a[0].col مرتبه تکرار می شود که a[0].col حاوی تعداد ستون های ماتریس اولیه است. علاوه بر این ، یک یک تکرار حلقه داخلی for به زمانی برابر با a[0].value نیاز دارد که در اینجا ، a[0].value تعداد عناصر در ماتریس اولیه است. بنابراین زمان کلی برای حلقه های تودرتوی for برابر با حاصل ضرب ستون ها در عناصر(columns.elements) می باشد. بنابراین زمان اجرا به صورت (columns.elements) خواهد بود.

آرایه های چند بعدی

❖ دو راه متداول برای نمایش آرایه های چند بعدی وجود دارد:
- روش سطری
- روش ستونی

❖ در روش سطری آرایه های چند بعدی را به وسیله سطرهای آن ذخیره می کنیم.

مثال آرایه دو بعدی $A[upper_0][upper_1]$ نشان می دهد که دارای $upper_0$ سطر است $(row_0, row_1, ..., row_{upper0-1})$

به نحوی که هر سطر شامل $upper_1$ عنصر می باشد.

آرایه های چند بعدی

مثال

اگر α آدرس $A[0][0]$ باشد، بنابراین آدرس $A[i][0]$ ، $i.\ Upper + \alpha$ خواهد بود.

برای نمایش یک آرایه سه بعدی $A[upper_0][upper_1][upper_2]$ آن را به عنوان $upper_0$ آرایه دو بعدی با ابعاد $upper_1 \times upper_2$ در نظر می گیریم.