
madsage
IRan Education
Research
NETwork
(IRERNET)
شبکه آموزشی-پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
مادسیج
شبکه آموزشی -پژوهشی ایران
madsg.com
مادسیج

فصل 7

آرایه‌ها

هدف کلی
آشنایی با قابلیتهای متعدد آرایه‌ها

هدفهای رفتاری
از دانشجو انتظار می‌رود، پس از خواندن این فصل،
1. با مفهوم آرایه آشنا شود.
2. بردار و ماتریس را بشناسد.
3. نحوهٔ تعریف آرایه‌های یک‌بعدی را بیان کند.
4. کلاس حافظه در آرایه‌ها و نحوهٔ مقداردهی اولیهٔ آنها را بشناسد.
5. چگونگی تعریف آرایه‌های چندبعدی را بداند.
6. با نحوهٔ انتقال آرایه به تابع آشنا شود.
7. با رشته‌ها، ثابت رشته‌ای، و متغیر رشته‌ای آشنایی پیدا کند.
8. هدف از به کاربردن آرایه‌ها در مرتب‌سازی و روشهای آن را بداند.
9. انواع رشته‌های جستجو را نام ببرد.
10. با توابع کتابخانه‌ای strcpy، strcat، strlen، strcmp، و strcmoi آشنا شود.

مقدمه
آرايه مجموعه عناصري است كه ويژگيها و صفات يكسان ی دارند. به عبارت ديگر آرايه فضاي پيوسته ای از حافظهٔ اصلي كامپيوتر است كه مي‌تواند چندين مقدار را در خود جاي دهد. همهٔ عناصر يك آرايه از يك نوع‌اند و بااندیس مشخص مي‌شوند.
از نظر رياضي معمولاً آرايه هاي يك بعدي را بردار و آرايه هاي دوبعدي را ماتريس نامند . همچنين به طريق مشابه مي توان آرايه هاي چندبعدي را تعريف كرد.

تعریف آرایه‌ها
در زبان C، آرایه‌ها به شکل متغیرهاي معمولي تعريف مي‌شوند با اين تفاوت كه نام آرايه بايد با مشخصهٔ اندازه همراه باشد.

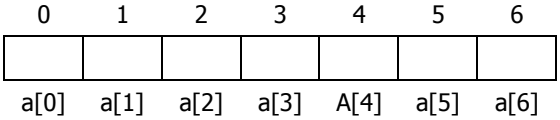
آرایهٔ یک‌بعدی
آرايهٔ يك بعدي به صورت زير تعريف مي‌شود.

type array-name [array-size] ;

كه در آن array-name نام آرايه است كه از قانون نامگذاري متغيرها پيروي مي كند، array-size بزرگي و يا تعداد عناصر آرايه است و type نيز نوع عناصر آن را مشخص مي‌كند. براي مثال اگر آرايهٔ a داراي 7 عنصر از نوع int باشد، به‌اين صورت معرفي مي‌شود.

int a[7] ;

و خانه‌هاي اختصاص داده شده به آن به صورت متوالي و به شكل زير خواهد بود.



همان طور كه مي‌بينيد شمارهٔ خانه‌ها از صفر تا شش است . به عبارت ديگر حد پايين آن برابر صفر و حد بالاي آن يك واحد از طول يا بزرگي آرايه كمتر خواهد بود كه در مثال مزبور، حد بالاي آن برابر 6 است.
روش برنامه‌نويسي خوب آن است كه اندازهٔ آرايه به صورت ثابت سمبوليك تعريف شود. از آنجا كه با تغيير مقدار ثابت سمبوليك اندازهٔ آرايه به راحتي تغيير مي‌كند، اين عمل تغيير برنامه‌اي را كه از آرايه سود مي‌برد ساده‌تر مي‌سازد.

ϖ **مثال 7ـ1** به دستورهای زیر توجه کنید.

```
#define size 50
int A[size] ;
```

در اينجا طول آرايه به صورت غيرمستقيم و با استفاده از دستور define مشخص شده است. كلاس حافظه ممكن است خودكار، ايستا، يا خارجي باشد، اما نمي‌تواند ثبات تعريف شود. بنابراين در حالت كلي مي‌توان آرايه‌اي يك‌بعدي را به صورت زير تعريف كرد.

storage-class data-type array-name [expression] ;

كه در آن expression ممكن است عدد صحيح يا متغير از نوع int و يا عبارت سادهٔ محاسباتي باشد كه از تركيب مقادير عددي صحيح و متغيرهايي از نوع int با استفاده از عملگرهاي محاسباتي مانند + و - تشكيل شده است. نتيجهٔ اين عبارت يك عدد صحيح خواهد بود كه معرف

بزرگي آرايه است. متعارف آن است که بزرگي آرايه به صورت عدد صحيح و يا متغيري از نوع عدد صحيح تعيين گردد . واضح است که اگر بزرگي آرايه به صورت متغير از نوع int بيان گردد، بايد مقدار آن هنگام تعريف عناصر آن تشخيص داده شود . براي آرايه‌هايي که درون يک تابع يا بلوک تعريف مي‌گردند سطح ذخيره‌سازي خودکار و براي آرايه‌هايي که بيرون از تابع تعريف مي‌گردند سطح ذخيره‌سازي خارجي پيش فرض خواهد بود.

ϖ

ϖ **مثال 2.7** به نمونه‌هايي از تعريف چند آرايۀ يك بعدي توجه کنيد.

```
int A[5] ;
float B[25] ;
static float C[15] ;
double x1[10] ;
char str[80] ;
```

در اين اعلان آرايۀ A از نوع int با 5 عنصر و آرايۀ B از نوع float با 25 عنصر و آرايه C از نوع float با 15 عنصر و از لحاظ کلاس حافظه نوع ايستا تعريف شده است. همچنين آرايۀ x1 از نوع double و آرايۀ str از نوع char با 80 عنصر تعريف شده است. (آرايه‌هاي کاراکتري معمولاً براي نمايش رشته‌ها به کار مي روند.)

ϖ

مراجعه به عناصر آرايه

وقتي که آرايه اي را تعريف مي کنيم، کامپايلر مجموعه ای حافظه به صورت پيوسته يا متوالي براي آن در نظر مي گيرد. وقتي که آرايه اي ايجاد شد، براي مراجعه به هر عنصر آن کافي است پس از نام آرايه، شمار ۀ عنصر مورد نظر را در درون يك زوج کروشه قرار دهيم . همچنين در زبان C، انديس آرايه از صفر آغاز مي‌گردد.

ϖ **مثال 3.7** اگر A نام آرايه‌اي باشد که از پيش تعريف شده و مقاديري نيز به آن اختصاص داد ه شده باشد در اين صورت دستور k = A[2] ; يك کپي از محتواي خانۀ شمارۀ 2 آرايه (يعني سومين عنصر آرايه) را به متغير k نسبت مي‌دهد.

يادآور مي‌شويم که نامگذاري آرايه ها از قانون نامگذاري متغيرها تبعيت مي کند و نوع عناصر آن نيز مانند متغيرهاي معمولي ممکن است int، float، char و جز آن باشد.

ϖ

کلاسهای حافظه درآرايه (و نحوۀ مقداردهي اولۀ آنها)

گفتيم که آرايه‌ها از نظر کلاس حافظه ممکن است خودکار، ايستا يا خارجي تعريف شوند، ولي نمي توانند ثبات تعريف شوند. آرايه‌هاي از نوع خودکار، برخلاف متغيرهاي خودکار ممکن است هنگام تعريف، مقدار اوليه بپذيرند.

حال باتوجه به اين توضيح‌ها مي‌توان شکل کلي مقداردهي اوليه به آرايه‌ها را به صورت زير بيان کرد.

```
storage-class data-type array-name [size] = {value1 , value2 , … valuem} ;
```

که در آن value1، value2...valuem به ترتيب مقدار اولۀ اولين، دومين،... و m اُمين عنصر آرايه را مشخص مي‌کنند.

فرض کنيد که آرايه‌هاي a، b و c به صورت زير تعريف شده‌اند.

```
int a[7] = {1 , 2 , 3 , 4 , 5 , 6 , 7} ;
static float b[5] = {2.5 , -3.5 , 1.25 , 12.5 , 3.14} ;
char c[3] = {'a' , 'b' ,'c'} ;
```

ملاحظه مي‌کنيد که آرايۀ b ازنظر کلاس حافظۀ ايستا معرفي شده است، ولي کلاس حافظۀ دو آرايۀ a و c به طور صريح بيان نشده است. بنابراين بايد فرض کرد که هر دوي آنها خارجي‌اند. پس مقادير عناصر سه آرايۀ مزبور به صورت زير خواهد بود.

```
a[0] = 1 b [0] = 2.5        c (0) = a
a[1] = 2 b [1] = 3.5        c (1) = b
a[2] = 3 b [2] = 1.25       c (2) = c
a[3] = 4 b [3] = 12.5
a[4] = 5 b [4] = 3.14
a[5] = 6
a[6] = 7
```

اگر آرايه‌اي به صورت مقداردهي اوليه تعريف گردد، نياز نيست بزرگي يا اندازۀ آن را مشخص کنيم.

ϖ **مثال 4.7** به اعلان آرايۀ زير توجه کنيد.

```
int digit[ ] = {1 , 2 , 3 , 4 , 5} ;
```

که عناصر آن به اين صورت خواهد بود.

```
digit[0] = 1
digit[1] = 2
digit[2] = 3
digit[3] = 4
digit[4] = 5
```

در مثالهاي ذکر شده، آرايه هايي که مقادير اوليه گرفته اند و کلاس حافظۀ آنها به طور صريح مشخص نشده از نوع خارجي فرض شده اند (يعني از نوع خودکار نيستند).

ϖ

ϖ **مثال 5.7** برنامۀ زير نمرۀ امتحاني 25 نفر دانشجويان کلاسي را در درس رياضي مي خواند و تعداد دانشجويان مردود و همچنين تعداد دانشجوياني را که نمرۀ امتحاني آنان کمتر از معدل کلاس است تعيين و در خروجي چاپ مي کند.

```
# include<stdio.h>
main ()
  {
```



```
    int i , f = 0 , p = 0 ;
```

```
    float a[25] , average , sum = 0 ;
    for (i = 0 ; i<25 ; ++ i)
      {
        scanf ("%f " , &a[i]) ;
        sum = sum + a[i] ;
        if (a[i]<10)
          f = f + 1 ;
      }
    average = sum / 25 ;
    for (i = 0 ; i<25 ; ++i)
      if (a[i] < average)
        p = p + 1 ;
    printf ("%f %d %d" , average , f , p) ;
  }
```

در اين برنامه f و p به‌ترتيب معرف تعداد دانشجويان مردود و دانشجوياني است كه نمرة آنان كمتر از معدل كلاسي است و در آغاز مقدار اولية صفر داده شده است. در ضمن يادآور مي‌شويم كه اين برنامه را به طور متعارف نمي‌توانيم بدون استفاده از آرايه بنويسيم، زيرا قبل از اينكه نمرة امتحاني همة دانشجويان خوانده شود و معدل كلاس محاسبه گردد، نمي توان تعيين كرد كه آيا نمرة دانشجوي مورد نظر از معدل كلاس كمتر است يا نه. بنابراين چنانچه نمره‌های دانشجويان را با متغير ی ساده معرفي كنيم، هر بار كه نمرة دانشجوي جديدي خوانده مي شود، در همان حافظه مي‌نشيند. لذا نمرة دانشجوي قبلي پاك مي‌گردد. پس اگر معدل كلاس را بدون استفاده از آرايه حساب كنيم، در پايان خواندن نمرة دانشجويان به حافظة كامپيوتر، فقط نمرة نفر آخر را در حافظه خواهيم داشت و درنتيجه در مورد دانشجويان اول تا بيست و چهارم نمي توانيم تعيين كنيم كه نمرة كدام يك از آنها كمتر از معدل كلاس است. پس در اين مثال استفاده از آرايه الزامي است. درنتيجة آن 25 نمره به 25 آدرس مختلف خوانده مي‌شود. همچنين بايد توجه كنيم كه چون انديس از صفر شروع مي شود، شمارة انديسها از صفر تا بيست و چهار خواهد بود.

ϖ

آرايه‌هاي چند بعدي

آرايه‌هاي چندبعدي نيز مشابه آرايه هاي يك‌بعدي تعريف مي گردند، با اين تفاوت كه طول يا بزرگي هر بعد آرايه در داخل يك زوج كروشه مشخص مي‌گردد. از اين رو آرايهٔ دوبعدي دو جفت كروشه و آرايهٔ سه بعدي سه جفت كروشه دارد.

بنابراين شكل كلي تعريف آرايهٔ n بعدي به صورت زير خواهد بود.

```
storage-class data-type array-name[size1][size2]... [sizen] ;
```

كه در آن size 1 , size2 ,..., sizen به‌ترتيب بزرگي بعد يكم تا n اُم آرايه است.

براي مثال يك آرايهٔ دوبعدي 4 × 3 از نوع int را مي‌توان به صورت زير معرفي كرد.

```
int a[3][4] ;
```

در شكل 1.7 آرايهٔ دوبعدي m × n (شامل m سطر و n ستون) را مي‌بينيد.

به طريق مشابه مي‌توان آرايه‌هاي 3 بعدي يا بيشتر را تعريف كرد، مانند مثالهاي زير.

```
int a[3][4][5] ;
float b[2][3][8][5] ;
char page[2][5][10] ;
```

	ستون 1	ستون 2	ستون 3		ستون n
سطر 1	a[0][0]	a[0][1]	a[0][2]	...	a[0][n -1]
سطر 2	a[1][0]	a[1][1]	a[1][2]	...	a[1][n -1]
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.
سطر m	a[m -1][0]	a[m -1][1]	a[m - 1][2]	...	a[m -1][n -1]

شكل 1.7 آرايهٔ دوبعدی m × n

همچنين مي‌توان هرگام تعريف آرايه هاي دوبعدي يا بالاتر به آنها مقدار اوليه نسبت داد كه البته بايد آرايهٔ مورد نظر از كلاس حافظهٔ ايستا يا خارجي باشد.

ϖ **مثال 6.7** دستور زير را در نظر بگيريد.

```
int array[3][4] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12} ;
```

در اينجا فرض بر اين است كه آرايه از كلاس حافظهٔ خارجي است. آرايهٔ مزبور را مي‌توان جدولي تصور كرد كه داراي 3 سطر و 4 ستون (4 عنصر در هر سطر) است. مقادير اوليه نيز به ترتيب به عناصر سطرها اختصاص مي يابد (يعني انديس يا زيرنويس سمت راست سريع تر حركت مي‌كند). بنابراين مقادير عناصر آرايهٔ مزبور به صورت زير خواهد بود.

```
array[0][0] = 1   array[0][1] = 2   array[0][2] = 3   array[0][3] = 4
array[1][0] = 5   array[1][1] = 6   array[1][2] = 7   array[1][3] = 8
array[2][0] = 9   array[2][1] = 10  array[2][2] = 11  array[2][3] = 12
```

توجه داشته باشيد كه عملكرد يا محدودهٔ تغييرات انديس اول (انديس سمت چپ) از صفر تا 2 و انديس دوم (انديس سمت راست) از صفر تا 3 است.

ϖ

نحوهٔ اختصاص مقادير اوليه به عناصر آرايه را مي توان با دسته بندي كردن آنها در داخل زوج آكولاد تغيير داد . براى مثال در آرايهٔ دوبعدي

مقادير داخل هر ز وج آكولاد دروني به ترتيب به عناصر يكي از سطرها نسبت داده خواهد شد . اگر تعداد مقادير موجود در درون هر زوج آكولاد دروني كمتر از تعداد عناصر سطر متناظر آن باشد ، به بقية عناصر سطر مزبور مقدار صفر نسبت داده خواهد شد . براي مثال، نتيجة عملكرد دستور زير

```
int array[3][4] = {
 {1 , 2 , 3 , 4} ,
 {5 , 6 , 7 , 8} ,
 {9 , 10 , 11 , 12}
 } ;
```

با مثال 6.7 يكسان خواهد بود.
حال دستور زير را درنظر بگيريد.

```
int array[3][4] = {
 {1 , 2 , 3} ,
 {4, 5 , 6} ,
 {7, 8 , 9}
 } ;
```

به عناصر ستون چهارم مقاديري نسبت داده نشده است . لذا مقادير آنها برابر صفر خواهد شد ؛ يعني مقادير عناصر آراية مورد نظر ب ه صورت زير خواهد بود.

```
array[0][0] = 1   array[0][1] = 2   array[0][2] = 3   array[0][3] = 0
array[1][0] = 4   array[1][1] = 5   array[1][2] = 6   array[1][3] = 0
array[2][0] = 7   array[2][1] = 8   array[2][2] = 9   array[2][3] = 0
```

درحالي كه اگر آراية مزبور را بـه صورت ; (int array[3][4] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 تعريف كنيم، مقادير نسبت داده شده به عناصر آراي مورد نظر به صورت خواهد بود.

```
array[0][0] = 1   array[0][1] = 2   array[0][2] = 3   array[0][3] = 4
array[1][0] = 5   array[1][1] = 6   array[1][2] = 7   array[1][3] = 8
array[2][0] = 9   array[2][1] = 0   array[2][2] = 0   array[2][3] = 0
```

انتقال آرايه به تابع

در زبان C، وقتي كه نام آرايه بـه عنوان آرگومان تابع ظاهر مي شود، آدرس اولين عنصر آرايه تعبير مي گردد. بنابراين هنگامي كه آرايه اي را به عنوان آرگومان به تابع گذر يا انتقال مي دهيم، تمامي آرايه به آن تابع انتقال مي يابد. اين روش انتقال آرگومان به تابع، با آنچه در مورد انتقال متغيرهاي معمولي به تابع در فصل مربوط به توابع بيان شد و فراخواني با مقدار ناميديم متفاوت است. روش فراخواني جديد را فراخواني با آدرس يا فراخواني با ارجاع نامند. در اين روش به جاي كپي داده‌ها، آدرس آنها به تابع انتقال مي‌يابد. تشريح كامل اين روش را در فصل 8 بيان مي‌كنيم.

براي گذر دادن آرايه اى به تابع بايد فقط نام آن بدون كروشه و بدون انديس، ب ه عنوان آرگومان واقعي ، در فراخواني تابع ظاهر گردد . در تعريف تابع نيز بايد آرگومان فرمال متناظر آن به همان طريق نوشته شود و ب ه عنوان آرايه تعريف گردد . بدين طريق كه نام آرايه همراه با يك زوج كروشة بدون انديس نوشته شود و نوع داده آن نيز مشخص گردد.

مثال 7.7 قطعه برنامة زير نحوة فرستادن يا گذردادن آرايه را به تابع نشان مي‌دهد.

```
main()
  {
    int n ;         /* variable declaration */
    float avg ;     /* variable declaration */
    float list [100] ;        /* array definition */
    float average () ;        /* function declaration */
    ....
    avg = average (n , list) ;
    ....
  }
    float average (a , x)          /* function definition */
    int a ;         /* formal argument declaration */
    float x[ ] ;    /* formal argument (array) declaration */
      {
        ....
      }
```

ملاحظه مي‌كنيد كه تابع فرعي average در داخل تابع اصلي فراخوانده شده است. اين تابع داراي دو آرگومان است: يكي متغير n كه نوع آن int و معرف تعداد عناصر آرايه است . ديگري آراية يك‌بعدي list كه نوع عناصر آن float است. همچنين مي‌بينيد كه در فراخواني تابع، آراية list به صورت متغير ساده ظاهر شده است.

در تعريف تابع نيز در سطر اول دو آرگومان فرمال را كه a و x ناميده شده‌اند مشاهده مي‌كنيد. سپس در دو سطر بعدي دو آرگومان مزبور به ترتيب به‌صورت int و آراية يك‌بعدي از نوع float توصيف شده‌اند. ملاحظه مي‌كنيد كه بين آرگومان واقعي n و آرگومان فرمال a تناظر وجود دارد. به طريق مشابه تناظري بين آرگومان واقعي list و آرگومان فرمال x وجود دارد. در واقع لازم نيست كه آرگومانهاي واقعي با آرگومانهاي فرمال همنام باشند و به همين دليل آرگومانهاي فرمال را متغيرهاي مجازي يا ساختگي نيز مي‌نامند. همچنين ملاحظه مي كنيد كه بزرگي آرايه x، در توصيف آرگومان فرمال، مشخص نشده است . همين طور مشاهده مي كنيد كه در تعريف تابع بـه صورت پيش‌نمونه يا prototype در تابع اصلي، پس از نام تابع فقط يك زوج پرانتز تهي به كار رفته است.

حال اگر در تعريف تابع فرعي، توصيف آرگومانهاي آن نيز در همان خط اول تابع، پس از ذكر نام تابع در درون زوج پرانتز انجام گيرد، بايد در تعريف پيش‌نمونه در تابع اصلي نيز اين تناظر محفوظ بماند ؛ يعني بايد پس از نام تابع آرگومانهاي آن نيز در درون زوج پرانتز توصيف گردند (برخلاف حالت قبل كه فقط يك زوج پرانتز تهي به كار می‌رفت). قطعه برنامة زير همان مثال قبلي را با اين شيوه نشان مي‌دهد.

main

```
  {
    int n ;         /* variable declaration */
    float avg ;    /* variable declaration */
    float list [100] ;         /* array definiation */
    float average (int , float[ ]) ;   /* function declaration */
   ....
    avg = average (n , list) ;
   ....
  }
float average (int a , float x [ ])/* function definition */
  {
   ....
   ....
  }
```

ϖ

در انتقال نام آرايه به تابع، آدرس اولين عنصر آرايه به تابع منتقل مي شود؛ يعني نام آرايه، آدرس اولين عنصر آرايه تفسير مي‌شود. وقتي كه تابع فراخواني مي شود، اين آدرس به آرگومان فرمال متناظر آن اختصاص مي يابد. پس آرگومان مزبور به عنوان اشاره‌گر به اولين عنصر آرايه برمي‌گردد. بنابراين هر عملي كه در تابع فرعي روي آرايه انجام گيرد، نتيج ة آن در تابع اصلي (يا تابع فراخواننده) نيز منعكس مي‌شود. به طوري كه بيان شد، اين شيوة فراخواني تابع را فراخواني با آدرس نامند.

وقتي كه در درون تابع فراخوانده شده به عنصر آرايه مراجعه مي شود، انديس عنصر مورد نظر به مقدار اشاره گر افزوده مي گردد تا به آدرس آن عنصر در درون آرايه دلالت نمايد . پس هر عنصري از آرايه به سادگي در درون تابع در دسترس قرار می‌گيرد و به طوري كه گفتيم هر عنصر آرايه كه در درون تابع تغيير يابد، اثر اين تغيير در تابع فراخواننده نيز منعكس خواهد شد .

ϖ **مثال 8.7** برنامة زير برنامة ساده‌ای را نشان مي‌دهد كه آرايه‌ای 3 عنصري را به تابع گذر مي دهد و مقادير عنصر آرايه در درون آن تابع تغيير مي‌يابد. مقادير عناصر آرايه در سه موقعيت از برنامه نوشته شده است تا اثر اين تغييرات را نشان دهد.

```
# include<stdio.h>
main ()
  {
    int count , a [3] ;       /* array definition */
    void modify (int a[ ]) ;            / * function declaration */
    printf("\n from main , before calling the function: \n") ;
    for (count = 0 ; count<=2 ; + + count)
      {
        a[count] = count + 1 ;
        printf ("a[%d] = %d\n" , count , a[count]) ;
      }
    modify(a) ;
    printf ("\n from main , after calling the function:  \n") ;
    for (count = 0 ; count<=2 ; ++ count)
        printf ("a[%d] = %d\n" , count , a[count]) ;
  }
void modify (int a[ ])    /* function definition */
  {
    int count ;
    printf ("\n from the function , after modifying the values: \n") ;
    for (count = 0 ; count< = 2 ; ++ count)
      {
        a[count] = -9 ;
        printf ("a[%d] = %d" , count , a [count]) ;
      }
    return ;
  }
```

در اولين حلقه‌اي كه در درون تابع اصلي ظاهرمي گردد، مقادير a [0] = 1 , a [1] = 2 , a [2] = 3 به عناصر آرايه نسبت داده مي شود و همين مقادير با دستورprintf نمايش می‌يابد. سپس، آراية مزبور به تابع modify گذر داده مي‌شود كه در درون تابع مزبور به هريك از عناصر آرايه مقدار 9- نسبت داده مي شود. سپس همين مقادير جديد در آن تابع چاپ مي گردد. بالاخره پس از برگشت كنترل از تابع فرعي به تابع اصلي، دوباره مقادير عناصر آرايه نمايش می‌يابد.

اگر برنامة مزبور اجرا گردد، خروجي زير را خواهيم داشت.

```
from main , before calling the function:
a [0] = 1
a [1] = 2
a [2] = 3
from the function , after modifying the values:
a [0] = -9
a [1] = -9
```



```
a [2] = -9
```

from main , after calling the function:
a [0] = -9
a [1] = -9
a [2] = -9

اين نتايج نشان مي‌دهد كه تغييرات اعمال شده با تابع modify روي آرايه، در تابع اصلي نيز منعكس شده است.

ϖ

آرايه‌ها و رشته‌ها

در زبان C رشته‌ها، آرايه‌اي از كاراكترها تعريف مي شوند به طوري كه هر كاراكتر رشته، درون يک عنصر از آرايه ذخيره مي گردد. هر رشته به كاراكتر null كه نشانة پايان است خاتمه مي‌يابد. ثابت رشته‌اي در داخل دوبل گيومه قرار می‌گيرد و وقتي كه ذخيره مي‌گردد، كاراكتر null به طور خودكار به انتهاي آن افزوده مي شود. در داخل يك برنامه، كاراكتر null در فرم escape sequence با '\0' نشان داده مي شود. ثابت رشته‌اي آرايه‌اي را معرفي مي‌كند كه محدوده يا انديس پايين آن صفر و محدوده يا انديس بالاي آن تعداد كاراكترهايي است كه در رشته وجود دارد .

ϖ **مثال 9.7** به رشتة زير توجه كنيد.

" payam noor university "

اين رشته آرايه‌ای 22 كاراكتري است (دو فضاي خالي بين كلمات و يک 0\ نيز كاراكتر شمرده مي‌شوند). تعريف اين رشته ممكن است در آرايه‌ای كاراكتري به شكل زير باشد.

char str[ ] = " payam noor university "

متغيرهاي رشته‌اي متغيرهايي‌اند كه مقادير آنها ممكن است يك رشته باشد. در واقع متغيرهاي رشته‌اي، آرايه‌هايي از نوع char اند.

ϖ

ϖ **مثال 10.7** برنامة زير 15 رشته (مثلاً اسامي 15 نفر) را مي‌خواند و آنها را در سطرهاي متوالي چاپ مي كند. فرض بر اين است كه طول هر رشته با درنظر گرفتن null character پايان رشته حداكثر 12 كاراكتر است.

```
# include<stdio.h>
main ()
  {
     char name[12] ;
     int i ;
     for (i = 1 , i < = 15 ; ++i)
        {
          scanf ("%s" , name) ;
          printf ("\n %s" , name) ;
        }
  }
```

ملاحظه مي‌كنيد كه متغير رشته اي name به صورت آرايه معرفي شده است . در ضمن در خواندن اطلاعات به كمك تابع scanf، فقط نام متغير، بدون اپراتور آدرس (يعني بدون علامت '&`) و بدون ذكر انديس آرايه ، به كار می‌رود، زيرا به طوري كه گفتيم، نام آرايه آدرس اولين خانه يا اولين عنصر آن است. همچنين هم در موقع خواندن مقدار براي name و همين طور در چاپ مقدار آن از كد فرمت %s استفاده شده است.

مي‌توان مجموعه‌ای از رشته‌ها را به صورت آرايه‌اي از رشته‌ها تعريف كرد. در اينجا چون خود رشته به تنهايي يك آرايه است، هر مجموعه از رشته‌ها مي‌توانند آرايه‌ای دوبعدي را تشكيل دهند كه اگر آنها را به صورت جدول مستطيل شكل تصور كنيم، هر سطر اين جدول معرف يكي از رشته‌ها خواهد بود.

ϖ

ϖ **مثال 11.7** قطعه برنامة زير اسامي 15 نفر را به آرايه رشته‌اي name مي‌خواند و آنها را در سطرهاي متوالي چاپ مي‌كند.

```
# include<stdio.h>
main ()
  {
     char name[15][12] ;
     int i ;
     for (i = 0 ; i<15 ; ++i)
        scanf ("%s" , &name[i]) ;
     for (i = 0 ; i<15 ; ++i)
        printf ("\n%s" , name[i]);
   }
```

ملاحظه كنيد كه در خواندن اطلاعات رشته اي به آرايه و در چاپ كردن آن فقط بعد اول آرايه به كار برده شده است؛ يعني در واقع اگر آرايه دو بعدي در اين مجموعه از رشته ها را، همان طور كه گفتيم، جدولی مستطيل شكل فرض كنيم كه در هر سطر آن يك رشته قرار دارد ، در خواندن اطلاعات هر بار سطر ی خوانده مي‌شود كه آدرس آن سطر و در نتيجه اپراتور آدرس نيز ب ه كار رفته است. به اين طريق مشابه در چاپ كردن آن نيز هر بار اطلاعات يك سطر كه شامل يك رشته است چاپ مي گردد.

ϖ

روشهاي مرتب سازي

يكي از كاربردهاي متداول آرايه ها، استفاده از آنها در روشهاي مرتب سازي است . چنانچه مجموعه‌اي از داده‌ها يا اطلاعات، براساس ويژگي يا نظم خاصي سازماندهي شوند، اين عمل را مرتب سازي گويند. در اين صورت پيدا كردن عنصر ی دلخواه در درون آن مجموعه، ساده تر و سريع‌تر انجام مي‌گيرد. بنابراين عمل مرتب كردن، به منظور سرعت بخشيدن به عمل جستجوست.

تعداد مقايسه‌ها و نيز تعداد جابه‌جايي عناصر، از عوامل اساسي در مورد سرعت مرتب سازيهايند. طبيعي است كه هر روش مرتب‌سازي كه سرعت بالا و منطق ساده تری داشته باشد و همچنين حافظ ة كمتري اشغال كند مطلوب تر است. بر اين اساس رو شهاي متعددي مطرح شده كه در ادامه چند نمونه را بررسي مي‌كنيم.

روش مرتب‌سازي حبابي

اين روش از نظر منطق، ساده ترين و از نظر كارآيي پايين ترين روش مرتب سازي اطلاعات به شمار مي‌رود. در اين روش ابتدا عنصر اول با عنصر دوم مقايسه مي‌شود. اگر عنصر اول بزرگ‌تر از دومي باشد، جاي آن دو تعويض مي شود. سپس اين عمل در مورد عنصر دوم با سوم و همين طور براي بقيه عناصر به صورت متوالي انجام مي گيرد؛ يعني، در پايان با عنصر n-1 اُم مقايسه مي شود كه اگر بزرگ تر از آن بود جايشان تعويض مي‌شود. وقتي كه اين عمل يك بار كامل شد، بزرگ ترين عنصر آرايه به آخر آرايه منتقل مي شود. حال اگر خانهٔ آخر را ناديده بگيريم و اين عمل را دوباره براي خانه ها يا عناصر 1 تا n-1 تكرار كنيم، اين بار بزرگ ترين عنصر خانه ها 1 تا n-1 (كه دومين عنصر بزرگ آرايه خواهد بود) به خانهٔ n-1 آرايه منتقل مي‌شود. اگر اين عمل را به صورت تكراري براي n-1 بار انجام دهيم و براي سرعت عمل بيشتر، در هر بار تكرار نيز خانهٔ آخر محدودهٔ جاري آرايه را ناديده بگيريم، يعني هر بار از طول ناحيهٔ مورد مقايسه يك واحد كاسته شود، در پايان، عناصر آرايه به صورت صعودي مرتب مي‌گردد. اگر بخواهيم كه عناصر آرايهٔ مورد نظر به‌صورت نزولي مرتب شود، بايد در مقايسهٔ دو عنصر متوالي a_i با a_{i+1}، چنانچه a_i كوچك‌تر از a_{i+1} باشد، جاي آن دو با يكديگر تعويض شود.

از آنجايي كه در اين روش مرتب سازي در هر بار تكرار، بزرگترين عنصر آرايه به سمت بالا يا انتهاي آرايه حركت مي‌كند (مانند حبابهاي هوا كه در آب جوش موجود است)، آن را مرتب سازي حبابي نامند.

ϖ **مثال 12.7** تابع زير آرايهٔ n عنصري A را به روش حبابي مرتب مي‌كند.

```
void BubbleSort (int A[ ] , int n)
  {
    int i , j , temp ;
    for (i =1 ; i<n ; + +i)
      for (j =0 ; j<n-i ; + +j)
        if (A[ j] > A[ j+1])
          {
            temp = A [ j] ;
            A[ j] = A[ j+1] ;
            A[ j+1] = temp ;
          }
  }
ϖ
```

روش مرتب سازي انتخابي

در اين روش مرتب سازي، شيوهٔ كار بدين صورت است كه محل بزرگ ترين عنصر را در درون آرايهٔ n عنصري مورد نظر مي‌يابيم و جاي آن را با عنصر آخر يعني a_n عوض مي‌كنيم؛ سپس بين عناصر a_1 تا a_{n-1} محل بزرگ‌ترين عنصر را (كه در واقع دومين عنصر بزرگ آرايهٔ اصلي خواهد بود) به‌دست مي‌آوريم و جاي آن را با عنصر a_{n-1} عوض مي‌كنيم. اين كار را n-1 بار تكرار مي‌كنيم. در اين صورت در تكرار آخر فقط دو عنصر a_1 و a_2 را خواهيم داشت كه بايد بزرگ‌ترين آن دو در خانهٔ a_2 قرار گيرد.

برتري اين روش نسبت به روش مرتب سازي حبابي آن است كه تعداد تعويض عناصر كمتر مي شود، زيرا در اين روش، در هر تكرار حداكثر يكبار جابه‌جايي انجام مي‌گيرد.

ϖ **مثال 13.7** تابع زير آرايهٔ n عنصري A را به روش انتخابي مرتب مي‌كند.

```
void SelectionSort (int a[ ] , int n)
  {
    int i , max , temp ;
    for (i = 1 ; i<n ; + + i)
      {
        max = 0 ;
        for (j =1 ; j<= n-i+1 ; + + j)
          if (A[ j] > A[max])
            max = j ;
          if (max != n-i)
            {
              temp = A[max] ;
              A[max] = A[n-i] ;
              A[n-i] = temp ;
            }
      }
  }
ϖ
```

روشهاي جستجو

يكي ديگراز كاربردهاي متداول آرايه ها، استفاده از آنها در روشهاي جس نجوست. هر گاه در داخل مجموعه اي از عناصر، دنبال عنصر خاصي بگرديم، اين عمل را جستجو كردن نامند. جستجو، در اغلب زمينه‌ها، بويژه در مورد بانكهاي اطلاعاتي و سيستمهاي تجاري، كاربرد زيادي دارند. شيوه‌هاي متعددي براي جستجو وجود دارد كه دو روش متداول آن را در ادامه بررسي مي‌كنيم.

جستجو به روش خطي

اين روش ساده‌ترين راه براي جستجو در آرايه يا جدول نامرتب است . براي اين كار عنصر مورد جستجو را به طور متوالي با عناصر اول تا n اُم آن جدول مقايسه مي‌كنيم. چنانچه در حين مقايسه، عنصر مزبور پيدا شد، شمارهٔ آن يادداشت مي‌شود و عمل جستجو خاتمه مي‌يابد. در غير اين صورت پيغام مناسب صادر مي‌شود. در اين روش رابطهٔ بين تعداد عناصر جدول و تعداد مقايسه‌ها به صورت معادلهٔ درجه اول خواهد بود.

ϖ **مثال 14.7** تابع زير عنصر x را در آرايهٔ n عنصري A به روش جستجوي خطي جستجو مي كند. اگر پيدا شد، انديس آن، و در غير اين

صورت مقدار صفر را برمي‌گرداند.

```
int LinearSearch (int A[ ] , int n , int x)
  {
    int i ;
    for (i = 0 ; i < n ; + + i)
      if (x = = A[i])
        return (i +1) ;
    return (0) ;
  }
```

جستجو به روش دودويي

روش ديگر براي جستجوي عنصري در داخل جدول يا آرايه روش دودويي است . اين روش در صورتي امكان‌پذير است كه عناصر جدول مورد نظر مرتب شده باشد. همچنين، در مقايسه با روش قبلي، از سرعت بيشتري برخوردار است.

در اين روش، عنصر ا ول و عنصر آخر جدول را با دو متغير مانند L و H نمايش مي‌دهيم و سپس عنصر مورد جستجو را با عنصر وسطي جدول يا m = (L + H) / 2 مقايسه مي كنيم. اگر مساوي بود ، عنصر مورد جستجو پيدا شده است . در غير اين صورت، چنانچه عنصر مورد جستجو از عنصر وسطي بزرگ تر باشد، L را مساوي m+l وگرنه H را مساوي m-1 قرار مي‌دهيم. سپس اين عمليات را باز هم تكرار مي كنيم؛ يعني باز هم عنصر وسطي جدول حاصل را به‌دست مي‌آوريم و عنصر مورد جستجو را با مقدار آن مقايسه مي كنيم. اين عمل تا موقعي كه شرط L ≤ H برقرار نباشد، ادامه مي‌يابد و پيغامي مبني بر عدم وجود عنصر مورد جستجو در داخل جدول مورد نظر چاپ مي گردد.

مثال 15.7 مراحل الگوريتم جستجو به روش دودويي براي ده عدد زير در جدول نمايش داده شده است.

65 , 141 , 192 , 205 , 218 , 389 , 424 , 500 , 538 , 567

جستجوي عدد 205				جستجوي عدد 567			
X	l	h	m	X	l	h	m
218	1	10	5	218	1	10	5
141	1	4	2	500	6	10	8
192	3	4	3	538	9	10	9
205	4	4	4	567	10	10	10

مثال 16.7 تابع زير در آرايهٔ مرتب شدهٔ n عنصري، به روش دودويي عنصر x را جستجو مي كند. اگر پيدا شد، انديس آن و در غير اين صورت مقدار صفر را برمي‌گرداند.

```
int BinarySearch (int A[ ] , int n , int x)
  {
    int middle , L , H ;
    L = 0 ;
    H = n-1 ;
    while (L <= H)
      {
        middle = (L+H)/2 ;
        if (x = = A[middle])
          return (middle +1) ;
        if (x >A[middle])
          L = middle +1 ;
        else
          H = middle -1 ;
      }
    return (0) ;
  }
```

توابع كتابخانه‌اي (در مورد رشته‌ها)

اغلب نسخه های زبان C، مجموعهٔ وسيعي از توابع كتابخانه اي در مورد عمليات روي رشته ها را پشتيباني مي كنند كه در مبحث توابع كتابخانه‌اي بحث كرديم. چند تابع بسيار متداول كه در نوشتن برنامه‌ها كاربرد زيادي دارند در جدول 7.1 آمده است.

جدول 7.1 چند تابع متداول در برنامه‌های كاربردی

نام تابع	عمل تابع
strcpy (s1 , s2)	رشتهٔ s2 را روي رشتهٔ s1 كپي مي‌كند.
strcat (s1 , s2)	رشتهٔ s2 را به دنبال رشتهٔ s1 ملحق (ضميمه) مي‌كند.

strlen (s)	طول رشتهٔ s را برمي‌گرداند.
strcmp (s1, s2)	رشتهٔ s1 را با رشتهٔ s2 مقايسه مي‌كند.

اگر بخواهيم درمورد مقايسهٔ دو رشته، تمايز بين حروف بزرگ و كوچك ناديده گرفته شود، تابع strcmp را به صورت strcmpi به كار مي‌بريم.
در ضمن يادآور مي شويم كه توابع كتابخانه اي مربوط به رشته ها در فايل string.h قرار دارند. پس اگر بخواهيم در بر نامه‌اي از اين تابع استفاده كنيم، بايد دستور include<string.h> # را نيز قبل از تابع main به كار بريم.
حال به چند مثال در مورد رشته‌ها توجه كنيد.

ϖ **مثال 17.7** برنامهٔ زير نحوهٔ استفاده و كاربرد چند تابع كتابخانه‌اي را نشان مي‌دهد.

```
# include<string.h>
# include<stdio.h>
main ()
   {
      char s1[80] , s2[80] , s3[80] ;
      gets (s1) ;
      gets (s2) ;
      printf ("\n Lengths:  %d %d\n" , strlen (s1) , strlen (s2)) ;
      if (!strcmp (s1 , s2))
         printf ("\n The strings are equal\n") ;
      strcpy (s1 , s3) ;
      strcat (s1 , s2) ;
      printf ("\n %s" , s3) ;
      printf ("\n %s" , s1) ;
   }
```

اين برنامه دو رشته از ورودي مي‌خواند و آن دو را به هم متصل مي كند.اگر اين برنامه را اجرا و براي دو رشتهٔ s1 و s2 كلمهٔ "computer science" را وارد كنيم، خروجي برنامهٔ مزبور به صورت زير خواهد بود.

```
Lengths:  16       16
The strings are equal
computer science
computer sciencecomputer science
```

ϖ

ϖ **مثال 18.7** برنامه‌اي بنويسيد كه اسامي n نفر دانشجو را بخواند، سپس آنها را به ترتيب الفبا مرتب و چاپ كند . n، حداكثر 15 است كه با اولين دستور ورودي خوانده مي‌شود.

```
# include<stdio.h>
# include<string.h>
main ()
   {
     int i , n , j ;
     char name [15][12] , temp [12] ;
     scanf ("%d" , &n) ;
     for (i=0 ; a<n ; ++i)
        scanf ("%s" , &name[i]) ;
      for (i=1 ; i<n ; ++i)
        for (j=0 ; j<n-i ; ++j)
           if (strcmp (name[j] , name [j+1]) > 0)
              {
                 strcpy (temp , name [j]) ;
                 strcpy (name[j] , name [j+1]) ;
                 strcpy (name[j+1] , temp) ;
              }
     for (i=0 ; i<n ; ++i)
        printf ("\n %s" , name[i]) ;
   }
```

در اين برنامه از آرايه‌هاي دوبعدي، روش مرتب سازي حبابي و چند تابع كتابخانه‌اي رشته‌اي استفاده شده است.

ϖ

خودآزمایی 7

1. ابااستفاده از آرايه برنامه‌اي بنويسيد كه جمله‌های اول تا دهم سري فيبوناچي را محاسبه و چاپ كند.
2. برنامه‌ای بنويسيد كه اعداد زوج 2 تا 20 را به عناصر آرايه‌ای 10 عنصري نسبت دهد و سپس شمارهٔ هر عنصر و مقدار متناظر آن را در دو ستون نشان دهد و چاپ كند.
3. برنامهٔ زير آرايه ای 10 عنصري را، هنگام تعريف آن، مقداردهي اوليه مي كند. سپس مجموع مقادير آن را محاسبه و چاپ مي كند. خروجي اين برنامه چيست؟

```
# include<stdio.h>
# define size 10
main()
```

```
{
   int i , total = 0 ;
   int A[size] = {10 , 9 , 8 , 7 , 6 , 5 , -5 , -6 , -7 , -8} ;
   for (i = 0 ; i<size ; i++)
      total += A[i] ;
   printf (" Sum = %d" , total) ;
}
```

4. برنامه‌اي بنويسيد كه عددي صحيح را دريافت كند و معادل آن را به شكل باينري (در مبناي 2) چاپ كند.
5. برنامه‌اي بنويسيد كه متني را از ورودي بخواند وحروف كوچک را به حروف بزرگ تبديل كند.
6. برنامه‌اي بنويسيد كه با استفاده از تابع كتابخانه اي rand كه اعداد تصادفي ايجاد مي كند، 6000 بار پرتاب تاس تخته نرد را شبيه سازي كند.
7. تفاوت بين فراخواني با مقدار و فراخواني با آدرس (يا فراخواني با ارجاع) در اين فصل به اختصار بيان شد . برنامه‌ای بنويسيد كه تفاوت انتقال (ارسال) تمامي آرايه به يك تابع (يعني درواقع ارسال آدرس آرايه به تابع) يا فراخواني با آدرس را با انتقال عنصري از آرايه به تابع، يعني فراخواني با مقدار، نشان مي دهد، به‌طوري كه در حلقة for اول مقادير آرايه چاپ گردد . سپس تابع modifyarray(a) فراخواني شود. در اين فراخواني، نام آرايه (يعني آدرس آغاز آرايه) به تابع انتقال يابد. تابع مزبور مقدار هريك از عناصر آرايه را دو برابر كند. پس از برگشت كنترل به تابع اصلي، مقادير آرايه چاپ گردد. نتيجة عملكرد تابع فرعي روي آرايه در تابع ا صلي نيز منعكس شود . سپس با فراخواني تابع modifyelement يك عنصر آرايه، به آن تابع انتقال يابد (كه درواقع فراخواني با مقدار است). تابع مزبور مقدار عنصر دريافتي را دوبرابر كند و نتيجه چاپ گردد. پس از برگشت مجدد كنترل به تابع اصلي، مقدار عنصر مزبور دوباره چاپ شود و نتيجة عملكرد تابع modifyelement در تابع اصلي منعكس نشود.
8. در علم آمار، سه پارامتر ميانگين، ميانه و مد مجموعه‌ای از مقادير به شكل زير تعريف مي‌شوند.
الف) ميانگين (mean) n مقدار $a_1 , a_2 , ... , a_n$ كه آن را ميانگين حسابي نيز نامند، برابر است با:

$$mean = \frac{a_1 + a_2 + ... + a_n}{n} = \frac{\sum_{i=1}^{n} a_i}{n}$$

ب) اگر عناصر را به صورت صعودي مرتب كنيم، چنانچه تعداد عناصر فرد باشد، مقدار عنصر وسطي را ميانه (media) نامند و چنانچه تعداد عناصر زوج باشد، نصف مجموع دو مقدار وسطي را ميانه نامند.
ج) مد (mode) عبارت از عنصري است كه بيشتر از بقيه تكرار شده باشد.
نتيجه عبارت است از يك پرسش از 99 نفر كه مقادير عددي صحيح بين 1 تا 10 است. برنامه‌اي بنويسيد كه 3 پارامتر ميانگين، ميانه و مد را محاسبه و چاپ كند و همچنين نمودار ميله‌اي يا هيستوگرام پاسخها را به صورت ستاره رسم كند.
9. برنامه‌اي بنويسيد كه 100 عدد را به حافظه بخواند . سپس عددي را از طريق ورودي دريافت كند و با فراخوانده شدن تابعي، عدد دريافتي در درون مجموعه اعداد جستجو شود . اگر پيدا شد، تابع مزبور شمارة آن را برگرداند . در غير اين صورت مقدار 1- برگرداند. سپس در تابع اصلي پيغام مناسبي مبني بر اينكه عدد مورد نظر پيدا شده است يا نه چاپ شود.
10. برنامه‌اي بنويسيد كه عناصر دو ماتريس a و b را كه داراي m سطر و n ستون باشند، به حافظه بخواند و سپس مجموع آن دو را براساس قانون جمع ماتريسها به دست آورد و چاپ كند. m و n حداكثر 8 اند كه با اولين دستور ورودي خوانده مي‌شوند.
11. برنامه‌اي بنويسيد كه عناصر دو ماتريس a و b را به حافظه بخواند و سپس حاصل ضرب آن دو را براساس قانون ضرب ماتريسها به دست آورد و نتيجه را به فرم ماتريس (آراية دوبعدي) چاپ كند. ماتريس a داراي m سطر و n ستون و ماتريس b، داراي n سطر و h ستون است. m و n و h حداكثر 7 اند كه از اولين دستور ورودي خوانده مي‌شوند.

12. برنامه‌اي بنويسيد كه يك سطر متن را با استفاده از تابع getchar و هر بار يك كاراكتر به يك آراية كاراكتري بخواند . سپس آراية كاراكتري مزبور را به صورت رشته چاپ كند.

13. تابعي بنويسيد كه طول رشته‌ای را به دست آورد و برگرداند.

14. تابعي بنويسيد كه بدون استفاده از تابع كتابخانه اي strcat، دو رشته را به هم ملحق كند (يعني رشتة دوم را به آخر رشتة اول ضميمه كند).

15. تابعي بنويسيد كه در درون يك رشته، زير رشتة ديگري را جستجو كند . اگر وجود داشت، محل آن (يعني از چندمين كاراكتر رشته اول شروع شده است) را برگرداند، در غير اين صورت مقدار 1- برگرداند.

16. تابعي بنويسيد كه در درون رشتة S1، از كاراكتر i اُم آن به طول j كاراكتر در رشته S2 كپي كند.

17. برنامه‌اي بنويسيد كه مجموعه ای از رشته ها را به حافظه بخواند و سپس با فراخوانده شدن تا بعي، رشته‌هاي مزبور به ترتيب الفبا مرتب كند. تعداد رشته‌ها حداكثر 10 رشته است كه پايان آن با كلمة "END" مشخص شده است.


شبکه آموزشی- پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
madsg.com
مادسیج
IRan Education & Research NETwork
(IRERNET)