
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 9

نوعهاي تعريف شده

هدف کلی
آشنایی با ساختار، typedef، اجتماع، و شمارش به منظور استفاده از مجموعه‌ای از داده‌هایی که ویژگیهای یکسان ندارند.

هدفهای رفتاری
از دانشجو انتظار می‌رود پس از خواندن این فصل،
1. با ساختار در نوع داده‌ای با صفات مختلف و تفاوت آن با رکورد آشنا شود.
2. با متداول‌ترین کاربرد ساختار، یعنی آرایه‌ای از ساختارها آشنایی بیابد.
3. نحوهٔ پردازش در ساختار را بشناسد.
4. نحوهٔ انتقال ساختار به تابع را بشناسد.
5. شکل کلی و کاربرد دستور typedef و مزیت آن را بر ساختار درک کند.
6. ساختار داده‌ها و اشاره‌گرها را بشناسد.
7. متغیر اشاره‌گر در عضو ساختار را بشناسد.
8. با اجتماع در نوع داده‌ای با صفات مختلف آشنا شود.
9. با شمارشی در نوع داده‌ای با صفات مختلف آشنا شود.

مقدمه
اغلب در کاربردهاي مختلف برنامه نويسي، با مجموعه اي از داده ها که ويژگيهاي يکسان ندارند مواج هيم و در اين حالت، آرايه ها قابل استفاده نيستند. در زبان C اين مشکل رفع شده است و اجازه مي‌دهد که کاربر چند نوع داده با توجه به نياز خود ايجاد کند که عبارت‌اند از:

الف) ساختار. عبارت است از دسته‌بندي متغيرهايي با صفات مختلف تحت يك نام.
ب) typedef. نام جديدي براي نوع (type) موجود ايجاد مي‌کند.
ج) اجتماع. با آن مي‌توان قسمتي از حافظه را براي استقرار دو يا چندين نوع متفاوت از داده ها تعريف کرد.
د) شمارشي. فهرستی از نشانه‌ها يا سمبولهاست که مي‌توان با مقادير صحيح شمارشي 1، 2، 3 و... به آنها مراجعه کرد.

ساختار[1]
ساختار مجموعه‌اي از متغيرهاست که با يك نام به آنها مراجعه مي‌شود. در واقع ساختار داده جديدي است که هر عنصر آن از نوع داده متفاوت است. اين شيوه وسيلهٔ ساده اي براي يکپارچه ساختن مجموعه اي از داده ها يا اطلاعات مربوط به هم در اختيار برنامه نويس قرار مي دهد. براي مثال فرض کنيد که از هر دانشجوي شمارهٔ دانشجويي، نام و نمرهٔ امتحاني وي با فرمت زير مورد نياز باشد.

st-no	name	grade

در اينجا نمي‌توان اين 3 فيلد را در آرايه‌ای 3 عنصري قرار داد، زيرا فيلد اول از نوع int، فيلد دوم از نوع رشته (آرايهٔ کاراکتري) و فيلد سوم از نوع float است؛ يعني صفات آنها يکسان نيستند. حال خواسته آن است که بتوان به اين مجموعهٔ 3 عنصري، تحت يك نام مراجعه کرد و اگر نياز باشد بتوان به عناصر يا فيلدهاي آن نيز مراجعه کرد. اين گونه مجموعه را در زبان پاسکال و کوبول رکورد و در زبان C ساختار تعريف مي‌کنند. اگر اين مجموع را rec بناميم، نحوهٔ تعريف آن در زبان C به صورت زير خواهد بود.

```
struct rec
  {
    int st-no ;
    char name[15] ;
    float grade ;
  } ;
```

در اينجا کلمهٔ کليدي struct براي تعريف داده از نوع ساختار به کار رفته که پس از آن نام انتخابي براي مجموعهٔ مورد نظر به کار برده شده و سپس در داخل يك زوج آکولاد، عناصر يا اعضاي مجموعهٔ مورد نظر تعريف شده اند و در پايان نيز سمبکولون `;` به عنوان پايان تعريف ساختار به کار رفته است. بنابراين در اين مثال کلمهٔ rec اين ساختار ويژه از داده‌ها را مشخص مي‌سازد.

حال مي‌توان متغيرهايي مانند x و z را از نوع ساختار مزبور تعريف کرد. براي اين کار کافي است بنويسيم

```
struct rec x , z ;
```

در واقع با اين تعريف، به کامپايلر مي‌گوييم که متغيرهاي x و z از نوع struct است که با نام rec تعريف شده است.
وقتي که ساختاری را تعريف مي کنيم، در واقع نوع پيچيده ای مرکب از عناصر ساختار را تعريف مي کنيم. بنابراين در مثال بالا کلمهٔ rec متغير نيست بلکه معرف نوع داده است، ولي کلمات x و z متغيرهايي‌اند که نوع آنها از نوع rec است.
مي‌توان x و z را به يکي از دو صورت زير نيز تعريف کرد.

روش اول **روش دوم**

[1]. structure

```
struct rec
  {
   int st-no ;
   char name[15] ;
   float grade ;
  } x , z ;
```

```
struct
  {
     int st-no ;
     char name[15] ;
     float grade ;
  } x , z ;
```

در هر دو روش بالا، اسامي متغيرهاي مورد نظر بلافاصله پس از تعريف ساختار و قبل از بستن آن با علامت `;´ كه پايان تعريف struct است آمده است. لذا ديگر نيازي به معرفي آنها با دستور struct rec نيست. در تعريف اول، نامي براي نوع داده انتخاب شده است. پس بعد از اين نيز هر جاي ديگر از برنامه مي توان متغيرهاي ديگري را از نوع struct rec تعريف كرد. اما در تعريف دوم نمي‌توان متغيري از اين نوع تعريف كرد. به هر حال شيوة بهتر آن است كه براي تعريف متغير ديگري از نوع ساختار مورد نظر در جاي ديگري از برنامه با محدوديت مواجه نشويم .

حال باتوجه به توضيحات بالا، چنانچه در حالت كلي نام ساختار 1 را با tag و اسامي اعضاي آن را با : member1 , member2 , ... , membe*rm* نشان دهيم، فرم كلي تركيب ساختار به صورت زير خواهد بود.

```
struct structure-name
  {
     type variable1-name ;
     type variable2-name ;
     ....
     ....
     type variablem-name ;
  } ;
```

```
struct tag
  {
     member1 ;
     member2 ;
     ....
     ....
     member*m* ;
  } ;
```

كه در آن struct كلمة كليدي براي تعريف داده از نوع ساختار و tag نام اين ساختار داده هاست. همچنين، member1 , member2 , ... , member*m* نيز معرف توصيف اعضاي ساختار مزبورند.

هر يك از اعضاي ساختار مي تواند متغير معمولي، اشاره گر، آرايه، يا حتي ساختار ديگري باشد . با اينكه ممكن است نام اعضاي يك ساختار با متغيرهاي ديگري كه در جايي ديگر از برنامه تعريف شده اند همنام باشد، گويا‌تر آن است كه در اين گونه موارد اسامي همنام به كار نبريم.

به اعضاي يك ساختار نمي‌توان كلاس حافظه اختصاص داد و همچنين نمي توان هنگام تعريف ساختار به اعضاي آن مقدار اوليه نسبت داد. وقتي تركيب ساختار يك بار تعريف گرديد، مي‌توان متغيرهايي از نوع ساختار مزبور به صورت زير توصيف كرد.

storage-class struct tag variable1 , variable 2 , ... , variable *m* ;

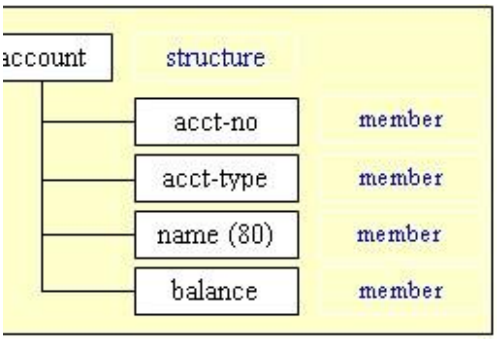
كه در آن storage-class مشخص‌كنندة كلاس حافظه است و به كار بردن آن اختياري است، ولي كلمة كليدي struct الزامي است و tag نيز نام ساختار است و متغيرهاي

variable1 , variable 2 , ... variable *m*

نيز متغيرهايي از نوع ساختار tag اند.

مثال 1.9 نمونه‌اي از تعريف ساختار، همراه با نماي ظاهري آن نشان داده شده است.

```
struct account {
   int acct-no ;
   char acct-type ;
   char name[80] ;
   float balance ;
}
```



نام ساختار، در اين مثال، account است و 4 فيلد يا عنصر دارد كه عبارت‌اند از:

- مقدار صحيح acct-no براى شماره حساب؛

- تك‌كاراكتر acct-type براى نوع حساب؛

- آرايه‌اى كاراكتري يا رشته‌اى name[80] براى نام صاحب حساب؛ و

- كميت يا مقدار balance براى مبلغ موجودي صاحب حساب.

حال مي‌توان متغيرهاي oldcustomer و newcustomer را از نوع ساختار مزبور به صورت زير تعريف كرد.

struct account oldcustomer , newcustomer ;

بنابراين، oldcustomer و newcustomer متغيرهايي از نوع ساختارند كه تركيب عناصر آنها با ساختار account مشخص شده است.

به طوري كه گفتيم مي‌توان توصيف ساختار را با متغيرهايي از نوع ساختار مزبور تركيب و آنها را يكجا به صورت زير تعريف كرد.

```
storage-class struct tag {
                          member1 ;
                          member2 ;
                          ...
                          memberm ;
                        } variable1 , variable2 ,... , variablem ;
```

در چنين موقعيتي، انتخاب نام براي ساختار، يعني به كار بردن tag، اختياري است.

ϖ

ϖ **مثال 2.9** توصيف زير، معادل دو توصيف مثال 9.1 است.

```
struct account {
                int acct-no ;
                char acct-type ;
                char name[80] ;
                float balance ;
               } oldcustomer , newcustomer ;
```

حال چون تعريف متغيرها با تعريف نوع ساختار تركيب شده است، مي توان نام ساختار، يعني account، را نيز به كار نبرد. البته اين شيوه توصيه نمي‌گردد.

در يك ساختار ممكن است عضوي از ساختار ديگر ی تعريف گردد. در چنين حالتي بايد تعريف ساختار دروني (يعني ساختار به كار رفته در درون ساختار ديگر) قبل از تعريف ساختار بيروني ظاهر گردد.

ϖ

ϖ **مثال 3.9** در برنامة C تعريفهاي زيرمفروض است كه به همراه نماي ظاهري آن نشان داده شده است.

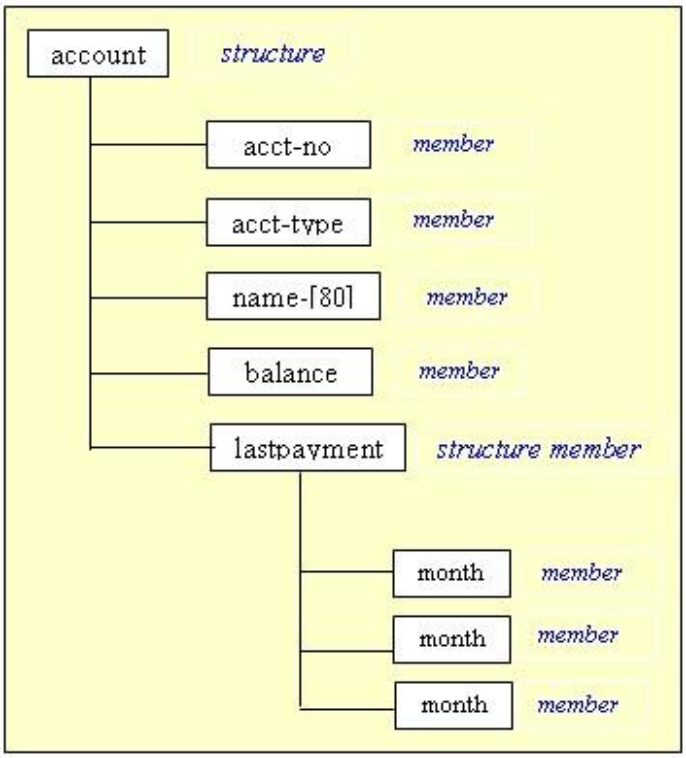
```
struct date {
        int month ;
        int day ;
        int year ;
} ;
struct account {
        int acct-no ;
        char acct-type ;
        char name[80] ;
        float balance ;
        struct date lastpayment ;
} oldcustomer , newcustomer;
```

ملاحظه مي‌كنيد كه ساختار account شامل ساختار ديگري، يعني date، است كه يكي از اعضا يا فيلدهاي خود است. در چنين موقعيتي بايد تعريف date قبل از تعريف account بيايد.

ϖ

اختصاص مقادير اوليه

به اعضا يا فيلدهاي متغير در ساختار مي‌توان، مشابه روشي كه در مورد عناصر آرايه ملاحظه



شکل 1.9 نمای ظاهری مثال 3.9

کردید، مقادیر اولیه نسبت داد . مقادیر اولیة مورد نظر، باید مشابه همان تر تیب تناظر اختصاص آنها به عناصر آرایه ظاهر گردند و در داخل زوج آکولاد قرار گیرند و با کاما از یکدیگر مجزا گردند. شکل کلي آن به صورت زیر است.

storage-class struct tag variable = {value1 , value2 , ... , valuem} ;

که در آن value1 , value2 ,...valuem معرف مقادیري است که باید به ترتیب به عناصر اول، دوم، ... و m اُم متغیر ساختار اختصاص یابد.

ϖ **مثال 4.9** قطعه برنامة زیر نحوة اختصاص مقادیر اولیه به اعضاي متغیر ساختار را نشان مي‌دهد.

```
struct date {
                int month ;
                int day ;
                int year ;
            } ;
struct account {
                int acct-no ;
                char acct-type ;
                char name[80] ;
                float balance ;
                struct date lastpayment ;
                } ;
static struct account customer = {12746 , `R´ , "payam noor" , 2986.50 , 5 , 24 , 75} ;
```

بنابراین، customer متغیر ساختاري ا ستا از نوع account است که به عناصر آن مقادیر اولیه اختصاص داده شده است . به اولین عنصر عدد صحیح 12746، به دومین عنصر کاراکتر `R´، به سومین عنصر رشتة " payam noor " و به چهارمین عنصر یا balance مقدار اعشاري 2986.50 اختصاص داده شده است . آخرین عنصر آن ساختار ی است که شامل سه مقدار صحیح (از چپ به راست، معرف ما ه، روز و سال) است. بنابراین به آخرین عضو customer مقادیر صحیح 75 , 24 , 5 اختصاص داده شده است.

ϖ

آرایه‌اي از ساختارها

در زبان C مي‌توان آرایه‌اي از ساختارها تعریف کرد ؛ یعني آرایه‌اي تعریف کرد که هر عنصر آن یك ساختار باشد . این شیوه، متداول ترین کاربرد ساختارهاست. براي توصیف آرایه‌اي از ساختارها، باید اول ساختار مورد نظر را توصیف کرد. سپس متغیري از نوع آرایه توصیف کرد که عناصر آن از نوع ساختار مزبور باشد.

ϖ **مثال 5.9** آرایه‌اي 100 عنصري به نام customer تعریف کنید که عناصر آن ساختاري از نوع مثال 4.9 یعني از نوع account باشد.

روش اول

روش دوم

struct date {

struct date {

```
        int month ;
        int day ;
        int year ;
} ;
struct account {
        int acct-no ;
        int acct-type ;
        char name[80] ;
        float balance ;
        struct date lastpayment ;
} customer[100] ;
```

```
        int month
        int day ;
        int year ;
} ;
struct account {
        int acct-no ;
        int acct-type ;
        char name[80] ;
        float balance ;
        struct date lastpayment ;
} ;
struct account customer[100] ;
```

در این توصیف، customer آرایه‌ای 100 عنصري است که هر عنصر آن ساختاري از نوع account است. درضمن هر دو روش توصیف بالا هم‌ارزند.

ϖ

به آرایه‌ای از ساختارها نیز مي‌توان مقادیر اولیه اختصاص داد. به‌خاطر داشته باشید که در اینجا عنصر آرایه ساختاری است که باید مقادیر اولیهٔ متناظر با هر عنصر به اعضاي آن اختصاص داده شود. مثال 6_9 نحوهٔ عمل را در این مورد نشان مي‌دهد.

ϖ **مثال 6_9** برنامه‌ای شامل توصیف زیر است.

```
struct date {
        char name[80] ;
        int month ;
        int day ;
        int year ;
} ;
static struct date birthday[ ] = { "Sara" , 12 , 30 , 73 ,
                                   "Hassan" , 5 , 13 , 66 ,
                                   "Dara" , 7 , 15 , 72 ,
                                   "Iraj" , 11 , 29 , 70 ,
                                   "Arash" , 2 , 4 , 77 ,
                                   "Susan" , 12 , 29 , 63 ,
                                   "Ahmad" , 4 , 12 , 69 } ;
```

در این مثال birthday آرایه‌اي از ساختارهاست که اندازهٔ آن مشخص نشده است. مقادیر اولیه اندازهٔ آرایه و حافظهٔ لازم را تعریف مي‌کند. لذا اندازهٔ آن برابر 7 خواهد بود، زیرا 7 سطر از مقادیر اولیه منظور شده است که از صفر تا 6 شماره‌گذاري خواهد شد.

ممکن است بعضي برنامه نویسان ترجیح دهند که هر مجموعه از ثابتها را با رعایت ترتیب آنها در داخل یك زوج آکولاد جداگانه قرار دهند . توصیف آرایهٔ مزبور با این شیوه، به صورت زیر خواهد بود.

```
static struct date birthday[ ] = {
                                   {"Sara" , 12 , 30 , 73} ;
                                   {"Hassan" , 5 , 13 , 66} ;
                                   {"Dara" , 7 , 15 , 72} ;
                                   {"Iraj" , 11 , 29 , 70} ;
                                   {"Arash" , 2 , 4 , 77} ;
                                   {"Susan" , 12 , 29 , 63} ;
                                   {"Ahmad" , 4 , 12 , 69}
} ;
```

ϖ

پردازش ساختار

اعضاي ساختار معمولاً به صورت مستقل و جدا از هم پردازش مي شوند، یعني هر عضو ساختار ، به عنوان هویت مستقل، به صورت جداگانه پردازش می‌شود. به هر عضو یا عنصر ساختار با استفاده از عملگر `.´ یا عملگر عضویت (که گاهي آن را عملگر period یا dot نیز نامند) دستیابي یا رجوع مي‌شود. با استفاده از این عملگر می‌توان به هر عنصر ساختار به صورت زیر دست یافت.

structure-name. element-name

یا

variable. member

اپراتور نقطه نسبت به سایر اپراتورها، حتي اپراتورهاي یکانی، تقدم بالایي دارد. براي مثال دو عبارت

+ + variable. member و + + (variable. member)

معادل هم‌اند؛ یعني اپراتور + + به عضو ساختار عمل خواهد کرد نه به تمام متغیر ساختار. به طریق مشابه، دو عبارت

&variable. member و &(variable. member)

معادل یکدیگرند. بنابراین هر دو عبارت مزبور به آدرس عضو ساختار دستوسی می‌یابند، نه به آدرس متغیر ساختار.

ϖ **مثال 6_7** برنامه‌اي بنویسید که عملیات زیر را انجام دهد.

1. مشخصات n دانشجو را که فرمت آن به صورت زیر است، به آرایه‌اي از ساختار بخواند و سپس سابقهٔ هر دانشجو را در سطر ی جدا چاپ کند. n حداکثر 25 است که از دستگاه ورودي استاندارد دریافت مي‌شود.
2. شمارهٔ دانشجویي را از ورو دي بخواند و در لیست دانشجویان جستجو کند . اگر وجود داشت بقیهٔ مشخصات وي چاپ شود، وگرنه

پیغام "not found" چاپ گردد.

3. سوابق دانشجويان برحسب شمارة دانشجويي به صورت صعودي مرتب شود و سپس سابقة هر دانشجو در سطری جدا چاپ گردد.

st-no	name		grade
	l-name	f-name	

متغيرهاي st-no، l-name، f-name و grade به ترتيب معرف شمارة دانشجويي، نام خانوادگي، اسم كوچك و نمرة امتحاني دانشجوست. درضمن ملاحظه مي‌كنيد كه سابقة هر دانشجو ساختار ی شامل 3 فيلد دارد: name،st-no ، grade كه در آن فيلد يا عضو name خودش ساختاری دوعضوي است. پس در اينجا با ساختارهاي تودرتو مواج هيم. اگر اسم ساختار اصلي را با strec نمايش دهيم، به هركدام از اعضاي آن مي‌توان به‌ترتيب با strec.st-no , strec.name , strec.grade رجوع كرد. همچنين به عناصر يا اعضاي name مي‌توان با strec.name.l-name , strec.name.f-name رجوع كرد . ملاحظه مي‌كنيد كه براي دستيابي به اعضاي دروني ساختار، با اپراتور نقطه به صورت تكراري يا تودرتو مواجهيم. اگر اسم ساختار اصلي را با strec نمايش دهيم به هركدام از اعضاي آن مي توان به‌ترتيب با strec.st-no , strec.name , strec.grade رجوع كرد . همچنين به عناصر يا اعضاي name مي‌توان با strec.name.l-name , strec.name.f-name رجوع كرد . ملاحظه مي‌كنيد كه براي دستيابي به اعضاي دروني ساختار، اپراتور نقطه به‌صورت تكراري يا تودرتو به كار رفته است.

برنامة مورد نظر به صورت زير است.

```
# include <stdio.h>
main ()
{ struct rec
        {
            int st-no ;
            struct name ;
                {
                    char l-name[15] ;
                    char f-name[15] ;
                } name1 ;
            float grade ;
        } ;
int i , j , n , s-n ;
struct rec strec[25] , temp ;
printf("enter the number of students \n") ;
scanf ("%d" , &n) ;
printf ("\n\n enter your data: \n") ;
for (i=0 ; i<n ; + + i)
{
    scanf ("%d" , &strec[i]. st-no) ;
    scanf ("%s %s" , strec[i].name.l-name , strec[i].name.f-name) ;
    scanf ("%f" , &strec[i].grade) ;
}
printf("display the records of students") ;
for (i=0 ; i<n ; + + i)
    printf("\n %d %s %s %f" , strec[i].st-no ,
        strec[i].name.l-name , strec[i].name.f-name , strec[i].grade) ;
printf ("\n enter a student number for search") ;
scanf ("%d" , &s-no) ;
printf ("\n search through list for indicated student") ;
for (i=0 ; i<n ; + + i)
    if (s-no = = strec [i].st-no)
        break ;
    if (i < n)
        printf ("\n %s %s %f" , strec[i].name.l-name ,strec.name.f-name , strec[i].grade) ;
    else
        printf("\n sort the student records in ascending order);
printf ("\n sort the student records in ascending order with respect to the student number") ;
for (i=0 ; i<n ; + +i)
    for (j=i+1 ; j<n ; + + j)
        if (strec[i]. st-no>strec[i]. st-no)
            {
                temp = strec[i] ;
                strec[i] = strec[j] ;
                strec[j] = temp ;
```

```
        }
printf ("\n display the sorted records of students") ;
for (i=0 ; i<n ; + + i)
      printf ("\n %d %s %s %f" , strec[i].st-no ,
      strec[i].name.l-name , strec[i].name.f-name , strec[i].grade) ;
}
```

ϖ

انتقال ساختار به تابع

مي‌توان اعضاي ساختار يا تمامي ساختار را به تابع گذر داد . راههاي مختلفي براي انتقال يا گذر اطلاعات از نوع ساختار به تابع و بالعكس وجود دارد. سازوكار انتقال برحسب اينكه بعضي اعضا يا تمامي ساختار را انتقال دهيم و همچنين برحسب گونه‌هاي مختلف C متفاوت است.

اعضا يا عناصر ساختار را مي توان هنگام فراخواني تابع، به عنوان آرگومان، به آن انتقال داد يا اينكه عضو خاصي از ساختار را با دستور return با تابعي به تابع فراخوانند ة آن برگرداند. براي انجام اين كار، با هريك از اعضاي ساختار مشابه متغير ی معمولي و تك‌مقدار برخورد مي‌شود.

ϖ **مثال 8.9** شكل كلي برنامة C در زير نشان داده شده است.

```
main ()
{
        typedef struct {    /* structure declaration */
         int month ;
         int day ;
         int year ;
 }date ;
        suruct {  /* structure declaration */
         int acct-no ;
         char acct-type ;
         char name[80] ;
         float balance ;
         date lastpayment ;
 }customer ;
 float adjust (char name[ ] , int acct-no , float balance) ;
        ...
customer.balance = adjust (customer.name , customer.acct-no , customer.balance) ;
        ...
}
float adjust (char name[ ] , int acct-no , float balance)
{
        float newbalance ;          /* local variable declaration */
        ...
        newbalance = … ;            /* adjust value of balance */
        ...
        return (newbalance) ;
}
```

اين برنامه شيوة انتقال اعضاي ساختار به تابع را نمايش مي‌دهد. در اينجا مقادير

customer.acct-no , customer.name , customer.balance

به تابع adjust گذر داده شده‌اند. در داخل تابع مزبور، مقداري كه به newbalance اختصاص مي‌يابد احتمالاً از انجام عمليات روي اطلاعا ت گذر داده شده به دست مي‌آيد. سپس اين مقدار به تابع اصلي برگردانده مي شود كه در آن به عضو customer.balance از متغير ساختار customer اختصاص مي‌يابد.

يادآوري. مي‌توان تابع پيش‌نمونة adjust در تابع ()main را به صورت زير نيز توصيف كرد.

```
float adjust (char[ ] , int , float) ;
```

به كمك اشاره گر به نوع ساختار مي توان تمامي ساختار را به عنوان آرگومان به تابع انتقال داد . اين شيوه، مشابه انتقال آرايه به تابع است. بديهي است كه اين روش روش انتقال با آدرس است. بنابراين نتيجة هر عضو ساختار كه در درون تابع فراخوانده شده ت غيير يابد، در تابع فراخواننده نيز تشخيص داده خواهد شد. لذا باز هم تشابه مستقيم بين اين انتقال و آنچه در مورد انتقال آرايه به تابع گفتيم مشاهده مي‌كنيد.

ϖ

ϖ **مثال 9.9** برنامة سادة زير را در نظر بگيريد.

```
typedef struct {
        char *name ;
        int acct-no ;
        char acct-type ;
        float balance ;
} record ;
```

```
main () / * transfer a structure-type pointer to a function */
 {
 void adjust (record *pt) ;
 static record customer = {"Nader , 3333 , `c´ , 33.33} ;
 printf (" %s %d %c %.2f \n" , customer.name , customer.acct-no ,
          customer.acct-type , customer.balance) ;
 adjust (&customer) ;
          printf (" %s %d %c %.2f \n" , customer.name , customer.acct-no ,
          customer.acct-type , customer.balance) ;
 }
void adjust (record *pt)
 {
          pt-> name = "Payam" ;
          pt-> acct-no = 9999 ;
          pt-> acct-type = `r´ ;
 pt-> balance = 99.99
          return ;
 }
```

این برنامه نحوة انتقال ساختار به تابع را با گذر دادن آدرس آن (يعني اشاره‌گر به ساختار) به تابع نمايش مي‌دهد.
customer از نوع ساختار است و از لحاظ کلاس حافظه نیز ا ستاست ، که به اعضاي آن مقادير اولية اختصاص داده شده است . ابتدا در تابع اصلي، اين مقادير اولية اعضاي customer با دستور printf نمايش داده مي شود. سپس ضمن فراخواني تابع adjust، آدرس ساختار به آن تابع گذر داده مي‌شود. در تابع مزبور به اعضاي customer مقادير ديگري نسبت داده مي‌شود. ملاحظه مي‌كنيد كه در اين تابع، متغير pt اشاره‌گر به ساختار تعريف شده است كه آدرس ساختار customer را دريافت مي كند. پس از برگشت كنترل به تابع اصلي، براي بار دوم مقدار اعضاي customer نمايش داده مي‌شود.
اگر برنامة مزبور اجرا شود، خروجي زير را مشاهده خواهيد كرد.

```
Nader 3333 c 33.33
Payam 9999 r 99.99
```

بازگشت اشاره‌گر به ساختار (با تابع)

تابع ممكن است اشاره‌گري به ساختار را به توابع فراخوانندة آن برگرداند. مثال 10.9 نحوة عمل را نمايش مي‌دهد.

مثال 10.9 برنامة زير با فراخواندن تابعي به نام search، آرايه‌اي از ساختارها كه شامل سابقة مشتري بانك است و همچنين شماره حساب مشتري را در اختيار تابع قرار مي‌دهد. تابع مزبور شماره حساب مورد نظر را در مجموع ة فهرست مشتريان جستجو مي‌كند. اگر چنين ركوردي وجود داشت آدرس آن وگرنه، NULL (صفر) برمي‌گرداند. سپس در تابع اصلي، درصورت وجود داشتن چنين ركوردي، بقية مشخصات صاحب شمارة حساب مذكور نمايش داده مي‌شود، در غير اين صورت پيغام مناسبي چاپ مي‌شود. عمل فراخواني تابع براي پيدا كردن ركورد خاص تا موقعي كه شمار ة حساب ارسالي به تابع مخالف صفر باشد ادامه مي‌يابد. سوابق دارندگان حساب به صورت مقدار اولية اي از آرايه اي از ساختارها نسبت داده شده است.

```
# include<stdio.h>
# define n 3
# define NULL 0
typedef stuct {
                    char *name ;
                    int acct-no ;
                    char acct-type ;
                   float balance ;
               } record ;
main ()
{
   static record customer[ ] = {
                                {"Nader" , 3333 ,'c' 33.33} ,
                                {"Payam" , 6666 ,'0' , 66.66} ,
                                {"Amir" , 9999 ,'d' , 99.99) ,
                                } ;                    /* array of structures */
   int acctn ;
   record *pt ;     /* pointer declaration */
   record *search (record table[ ] , int acctn) ;          /* function declaration */
   printf ("Customer Account Locator \n") ;
   printf ("To End , Enter 0 for the account number \n") ;
   printf ("\n Account no.: ") ;        /* enter first account number */
   scanf ("%d" , & acctn) ;
   while (acctn !=0)
       {
         pt = search (customer , acctn) ;     /* found a match */
```



```
         if (pt != NULL)
```

```
            {
               printf ("\n Name:  %s \n" , pt -> name) ;
               printf ("Accont no.:  %d \n" , pt -> acct-no) ;
               printf ("Account type:  %c \n" , pt -> acct-type) ;
               printf ("Balance:  %.2f \n" , pt -> balance) ;
            }
          else
             printf ("\n error-please try again\n") ;
          printf ("\n Account no.: ") ;          /* enter next account number*/
          scanf ("%d" , & acctn) ;
        }
 }
record *search (record table[ ] , int acctn)     / * function definition */
                    /* accept an array of structures and an account number ,
                    return a pointer to a particular structure (an array element)
                    if the account number matches a member of that structure */
{
   int count ;
   for (count = 0 ; count<n ; + + count)
     if (table [count].acct-no = = acctn)       /* found a match */
        return (&table[count]) ;       /* return pointer to array element */
   return (NULL) ;
}
```

اندازة آراية مورد نظر با ثابت سمبوليكي n بيان شده و مقدار آن 3 است؛ يعني براي سادگي كار، فقط 3 ركورد، براي نمونه ذخيره شده است. بديهي است كه در عمل، مقدار n خيلي بزرگ‌تر خواهد بود.

مي‌توان تمامي ساختار را به طور مستقيم به عنوان آرگومان به تابع انتقال داد يا با دستور return ساختار را به طور مستقيم از تابع برگرداند (برخلاف آرايه كه با تابع برگردانده نمی‌شود). چنين انتقال ساختاري به صورت مقدار خواهد بود . بنابراين اگر با تابع تغييراتي در اعضاي ساختار داده شود، نتيجة آن در تابع فراخواننده يا در خارج از تابع مذكو ر اعمال نخواهد شد. به هرحال اگر ساختار تغييريافته به نقطة فراخواني از برنامه برگردانده شود، تغييرات حاصل در اينجا شناخته خواهد شد.

ϖ

ϖ **مثال 11.9** برنامة زير را ملاحظه كنيد.

```
# include<stdio.h>
typedef struct {
                    char *name ;
                    int acct-no ;
                    char acct-type ;
                    float balance ;
                  } record ;
main () /* transfer a structure to a function */
{
  void adjust (record customer) ;    /* function declaration */
  static record customer = {"Nader" , 3333 , 'c' , 33.33} ;
  printf ("%s %d %c %.2f \n" , customer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
  abjust (customer) ;
  printf ("%s %d %c %.2f \n" , customer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
 }
void adjust (record cust)          /* function definition */
 {
   cust.name = "Payam" ;
   cust.acct-no = 9999 ;
   cust.acct-type ='r' ;
   cust.balance = 99.99 ;
   return ;
}
```

برنامة مزبور به جاي اشاره گر به نوع ساختار، تمامي ساختار (در واقع كپي ساختار) را به تابع مي فرستند. حال تابع adjust ساختاری را به عنوان آرگومان دريافت مي كند (برخلاف مثال 10.9 كه اشاره‌گر به ساختار را دريافت مي كرد). در اينجا چيزي از تابع به تابع main برگردانده نشده است.

اگر برنامة مزبور اجرا شود، خروجي زير حاصل خواهد شد.

```
Nader 3333 c 33.33
Nader 3333 c 33.33
```

در اينجا، نتيجة اجراي دستورهاي جايگذاري ، كه در تابع adjust به كار برده شده است، در تابع main شناخته نمي‌شود (يعني نتيجه در تابع اصلي منعكس نمي شود). دليل اين كار نيز واضح است ، زيرا انتقال ساختار customer از تابع main به تابع adjust با مقدار صورت گرفته

است.
حال فرض كنيد كه اين برنامه را به طريقي اصلاح كنيم كه تغييرات انجام شده در تابع adjust به تابع main برگردانده شود.

```
#include<stdio.h>
typedef struct {
                    char *name ;
                    int acct-no ;
                    char acct-type ;
                    float balance ;
                  } record ;
main ()          /* transfer a structure to a function and return the structure */
{
  record adjust (record customer) ; /* function declaration */
  static record customer = {"Nader" , 3333 ,'C ' , 33.33} ;
  printf (" %s %d %s %.2f \n" , customer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
  customer = adjust (customer) ;
  printf ("%s %d %.2f \n" , custo6mer.name , customer.acct-no ,
  customer.acct-type , customer.balance) ;
}
record adjust (record cust)       /* function definition */
 {
  cust.name = "Payam" ;
  cust.acct-no = 9999 ;
  cust.acct-type ='r' ;
  cust.balance = 99.99 ;
  return (cust) ;
 }
```

در اين برنامه، برخلاف مثال 10.9، تابع adjust ساختاری را كه همان ساختار تغييريافته در تابع مزبور است به تابع main برمي‌گرداند.
اجراي اين برنامه خروجي زير را ايجاد مي‌كند.

```
Nader 3333 c 33.33
Payam 9999 r 99.99
```

بنابراين تغييرات اعمال شده در تابع adjust در تابع main نيز منعكس شده است ، زيرا اين بار ساختار تغييريافته ب ه طور مستقيم به قسمت فراخواني برنامه برگردانده شده است.

ϖ

نوع داده كاربر

زبان برنامه‌نويسی C، دستور ويژه‌اي را معرفي مي‌كند كه اجازه مي‌دهد تا كاربران بتوانند نام جديدي براي نوع داده تعريف كنند . نام جديد معادل نوع داده مورد نظر خواهد بود . اين كار به كمك كلم ة كليدي typedef انجام مي‌گيرد. در واقع در اينجا، كلاس جديدي از داده ها ايجاد نمي‌گردد، بلكه براي نوع داده موجود نام جديدي تعريف مي گردد. پس از آنكه نام جديد براي نوع داده مورد نظر تعريف گرديد، م ي‌توان متغيرها، آرايه ها، ساختارها و غيره را برحسب نوع داده جديد توصيف كرد.
شكل كلي دستور typedef به صورت زير است.

```
typedef type name ;
```

يا

```
typedef type new-type ;
```

كه در آن type هريك از نوع داده هاي مجاز است و name يا new-type نيز نام جديد براي اين نوع اس ت. در واقع type يا يكي از نوع داده هاي استاندارد (مانند int، float، char و...) يا نوع داده تعريف شدة كاربر است كه قبلاً تعريف كرديم. به هرحال بايد توجه كرد كه نوع داده جديد فقط از نظر نام آن جديد است وگرنه همان طور كه گفتيم، كلاس جديدي از داده‌ها نيست.

ϖ **مثال 12.9** به دستور زير توجه كنيد.

```
typedef int age ;
```

در اين دستور، براي نوع داده int، نام جديد age انتخاب شده است. بنابراين از اين لحظه به بعد هر كجا نياز باشد كه داده اي به صورت int تعريف شود، مي‌توان آن را با نوع age تعريف كرد. بنابراين دو توصيف زير هم‌ارزند.

```
age a , b ;
int a , b ;
```

همين طور در دستورهاي

```
typedef float height[100] ;
height a , b ;
```

دستور اول، height را آرايه‌ای 100 عنصري از نوع float تعريف مي‌كند. دستور دوم، a و b را آرايه‌هاي 100 عنصري از نوع float توصيف مي‌كنند.
راه هم‌ارز ديگري براي بيان دو دستور بالا عبارت است از

```
typedef float height ;
height a[100] , b[100] ;
```

گرچه روش قبلي ساده‌تر است.
ويژگي typedef، خصوصاً در تعريف ساختارها، كار را ساده‌تر مي‌كند و كاربر را از نوشتن تكراري كلمة كليدي struct خلاص مي‌كند. در چند مثال قبل در مورد ساختارها نيز از اين دستور استفاده شد.

در شكل كلي، نوع ساختاری كه كاربر تعريف مي‌كند به صورت زير است.

```
typedef struct {
        member 1 ;
        member 2 ;
        ...
        member m ;
} new-type ;
```

که در آن new-type نوع ساختار تعریف شد ة کاربر است. حال متغیرهاي ساختار برحسب این نوع داده جدید (درواقع نام جدید براي نوع داده) تعریف می‌شوند.

ϖ

ϖ **مثال 13.9** دستورهاي زیر براي تعریف متغیرهاي ساختار ب ه کار می‌رود. در اینجا براي شرح ساختار از نوع داده تعریف شد ة کاربر استفاده شده‌است.

```
typedef struct {
                    int acct_no ;
                    char acct_type ;
                    char name[80] ;
                    float balance ;
                } Trecord ;
Trecord oldcustomer , newcustomer ;
```

Trecord از نوع دادة تعریف شدة کاربر توصیف شده است و متغیرهاي oldcustomer و newcustomer که ساختارند از نوع Trecord تعریف شده‌اند.

ϖ

ϖ **مثال 14.9** دستورهاي زیر نمونة دیگري از کاربرد دستور typedef را نمایش مي‌دهد.

```
typedef struct {
                    int day ;
                    int month ;
                    int year ;
                } date ;
typedef struct {
                    int st_no ;
                    char name[20] ;
                    date test ;
                } student ;
student Pnoor [100];
```

ϖ

ساختار داده‌ها و اشاره‌گرها

مي‌توان با به کار بردن عملگر آدرس، یعني &`´ ، به آدرس ابتداي ساختار دسترسي داشت (مشابه دستیابي به سایر آدرسها). براي مثال اگر variable معرف متغیری از نوع ساختار باشد، &variable آدرس ابتداي آن متغیر را نشان خواهد داد. همچنین مي‌توان متغیر اشاره‌گری به ساختار به صورت زیر تعریف کرد.

```
type *ptvar ;
```

که در آن type نوع داده است که دلالت بر ترکیب ساختار مي کند و ptvar نیز معرف نام متغیر اشاره گر است. حال مي‌توان آدرس آغاز متغیر ساختار را به صورت زیر به این اشاره‌گر نسبت داد.

```
ptvar = &variable ;
```

ϖ **مثال 15.9** تعریف زیر را در مورد ساختار در نظر بگیرید.

```
typedef struct {
        int acct_no ;
        char acct_type ;
        char name [80] ;
        float balance ;
} account ;
account customer , *pc ;
```

در این مثال، customer متغیر ساختار از نوع account و pc نیز متغیر اشاره‌گر است که آدرس متغیر ساختار از نوع account را در خود دارد. حال با دستور زیر آدرس آغاز customer به pc نسبت داده مي‌شود.

```
pc = &customer ;
```

توصیف متغیر اشاره‌گر را مي‌توان به صورت زیر با توصیف ساختار ترکیب کرد.

```
struct {
        member 1 ;
        member 2 ;
        ...
        member m ;
} variable , *ptvar ;
```

که در آن، variable، متغیر از نوع ساختار و ptvar نیز نام متغیر اشاره‌گر را معرفي مي‌کند.

ϖ

ϖ **مثال 16.9** توصیف زیر معادل دو توصیف معرفي شده در مثال 15.9 است.

```
struct {
        int acct_no ;
        char acct_type ;
        char name [80] ;
        float balance ;
} customer , *pc ;
```

حال مي‌توان، مشابه مثال قبل، آدرس آغاز customer را بلادستور زير به متغير اشاره‌گر pc نسبت داد.

```
pc = &customer ;
```

به عضو ساختار مي‌توان به صورت ptvar -> member دسترسي داشت كه در آن ptvar متغير اشاره‌گر به نوع ساختار مورد نظر است و مي‌توان آن را با عملگر نقطه، يعني `.´، قياس كرد. بنابراين عبارت ptvar -> member هم‌ارز عبارت variable. member است كه در آن variable متغيری از نوع ساختار است كه دربارهٔ آن بحث كرديم.

عملگر -> نيز مانند عملگر `.´ بالاترين تقدم را دارد. عملگر -> با عملگر نقطه تركيب می‌شود و بدين طريق عمل دستيابي به زيرعضو يا submember را در درون ساختار (يعني دستيابي به عضوي از ساختار كه خودش عضو ساختار ديگر است) فراهم می‌سازد. بنابراين مي‌توان به زيرعضوی به صورت زير دسترسي داشت.

```
ptvar ->member. submember
```

به چند روش مشابه، عملگر -> براي دستيابي به عنصری از آرايه كه خودش عضوي از ساختار است به كار می‌رود. اين عمل با دستوري مشابه زير ميسر است.

```
ptvar -> member [expression]
```

كه در آن expression مقدار صحيح غيرمنفي است و بر عنصر آرايه دلالت مي‌كند (يعني عنصر آرايه را مشخص مي‌سازد).

ϖ

ϖ **مثال 17.9** تعريف ساختار و متغيرهايي را به صورت زير درنظر بگيريد.

```
typedef struct {
        int month ;
        int day ;
        int year ;
} date ;
struct {
        int acct_no ;
        char acct_type ;
        char name [80] ;
        float balance ;
        date lastpayment ;
} customer , *pc = &customer ;
```

توجه داشته باشيد كه به متغير اشاره‌گر pc، آدرس آغاز متغير customer (كه از نوع ساختار است) به صورت مقدار اوليه نسبت داده شده است. به عبارت ديگر، متغير pc به customer اشاره مي‌كند.

حال اگر بخواهيم به شمارهٔ حساب مشتري دسترسي پيدا كنيم، اين كار به يكي از سه روش زير انجام می‌گيرد.

```
customer. acct_no
pc-> acct_no
(*pc). acct_no
```

وجود پرانتز در مورد عبارت آخري ضروري است، زيرا عملگر نقطه نسبت به عملگر `*´ تقدم بالاتري دارد و چنانچه پرانتز به كار نرود، كامپايلر نتيجهٔ اشتباه ايجاد مي‌كند، زيرا pc كه اشاره‌گر است، به صورت مستقيم با عملگر نقطه سازگار نيست.

به طريق مشابه مي‌توان به موجودي مشتري به يكي از سه روش زير دسترسي يافت.

```
customer. balance
pc-> balance
(*pc). balance
```

و به ماه آخرين پرداخت به يكي از دو روش زير.

```
customer. lastpayment. month
pc-> lastpayment. month
```

و بالاخره به نام مشتري به يكي از سه روش زير مي‌توان دسترسي داشت.

```
(*pc). lastpayment. month
pc->name
(*pc). name
```

همين طور به سومين كاراكتر نام مشتري به يكي از روشهاي زير دسترسي می‌يابيم.

```
customer. name[2]
*(customer. name + 2)
pc->name[2]
pc->(name+2)
(*pc). name[2]
*((*pc). name + 2)
```

يادآور مي‌شويم كه اعضاي ساختار مي‌تواند اشاره‌گر نيز باشد . اين روش در ساختارهايي مانند ليستهاي پيوندي، درختها، نمودارها و مشابه آن كاربرد زيادي دارد.

ϖ

عضو ساختار

متغير اشاره‌گر مي‌تواند عضو ساختار باشد. براي مثال مشخصات نفری با فرمت

نام	نام خانوادگي

را مي‌توان به صورت ساختار زير تعريف كرد.

```
struct names {
        char *lastname ;
        char *firstname ;
};
```

كه در اينجا، lastname و firstname اشاره‌گرهايي‌اند كه در واقع معرف دو رشته (يا دو آرايۀ كاراكتري) اند.

همچنين عضو ساختار ممكن است اشاره‌گري باشد كه آدرس عناصر ديگر يا آدرس ساختار ديگري را در خود داشته باشد يا آدرس ساختاري را از نوع خودش (يعني ساختاري كه در درون آن تعريف شده است) نگهداري كند.

در حالت كلي مي‌توان اين گونه ساختارها را به صورت زير تعريف كرد.

```
struct tag {
        member 1 ;
        member 2 ;
        ....
        ....
        ....
         struct tag &name ;
        } ;
```

كه در آن name متغيري از نوع اشاره‌گر است كه آدرس متغير ديگري از نوع ساختار به شكل tag در آن قرار مي‌گيرد. يكي از مهم‌ترين كاربردهاي روش بالا، در ايجاد ساختارهايي به نام ليست پيوندي و انجام عمليات يا پردازش روي آن است.

اجتماع[1]

اجتماع مانند ساختار داده هايي است كه از چند عضو تشكيل مي‌شود و هر عضو آن نوع داده اي منحصر به خود دارد. اجتماع محلي از حافظه است كه چندين متغير در آن قرار مي گيرند كه ممكن است نوع آنها نيز يكسان نباشد . در واقع اجتماع متغيري است كه امكان ذخيره كردن انواع مختلف داده در مكان مشتركي از حافظه را فراهم مي كند.

تعريف اجتماع مشابه تعريف ساختار است با اين تفاوت كه در اجتماع تمام اعضاي تشكيل‌دهندۀ آن فضايی را به صورت اشتراكي اشغال مي‌كنند (برخلاف ساختار كه هر عضو آن محل حافظه يا مكان خاص خودش را داراست)، از اين رو به منظور صرفه جويي در حافظه به كار گرفته مي‌شود. اين گونه ساختمان داده‌ها در كاربردهايي مفيدند كه اعضاي چندگانه از نوع مختلف دارند، ولي در هر زمان بايد فقط به يكي از اين اعضا مقداري نسبت داد. به هرحال كاربر بايد بداند كه هر لحظه چه نوع داده در حافظۀ مشترك مورد نظر ذخيره شده است.

در حالت كلي تركيب اجتماع ممكن است به صورت زير تعريف شود.

```
union tag {
        member 1 ;
        member 2 ;
        ...
        ...
        ...
        member m ;
} ;
```

كه در آن union كلمه‌ای كليدي است.

ϖ **مثال 18.9** به تعريف زير توجه كنيد.

```
union union_type {
        int i ;
        char ch ;
        } ;
```

اين تعريف نيز مشابه تعريف ساخ تار، موجب توصيف يا اعلان متغير نمي گردد، بلكه فقط تعريف نوع داده است . براي اعلان هر متغيري از نوع اجتماع مورد نظر، بايد نام آن را به دنبال تعريف union به كار برد يا به طور جداگانه آن را اعلان كرد. پس روش اعلان متغيرهايي از نوع اجتماع مشابه اعلان از نوع ساخ‌تار است.

ϖ

با توجه به مطالب بالا، متغير x به دو روش اعلان شده است كه نتيجۀ نهايي هر دو هم‌ارز است.

روش اول

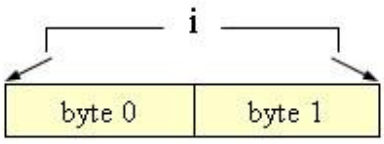
```
union tag {
    int i ;
    char ch ;
} ;
union tag x ;
```

روش دوم

```
union tag {
    int i ;
    char ch ;
} x ;
```

در اينجا، متغيرهاي i و ch كه به ترتيب از نوع int و char اند، حافظۀ مشترك دارند كه البته متغير i دو بايت و متغير ch يك بايت حافظه اشغال مي‌كند، ولي آدرس شروع آنها يكسان است (شكل 2.9).

[1]. union



شكل 2.9 نحوة اشغال حافظة مشترك با دو عضو اجتماع

وقتي كه اجتماع اعلام مي‌گردد، كامپايلر به طور خودكار متغيری كه بتواند بزرگ‌ترين عضو اجتماع را در خود جاي دهد ايجاد مي كند كه در مثال بالا بزرگي آن 2 بايت است.
باتوجه به مطالبی كه گفتيم، شكل كلي اعلان متغيرهايي از نوع اجتماع به صورت زير است.

storage _ class union tag variable1 ,variable 2 , ... , variable n ;

كه در آن storage_class گزينۀ اختياري است كه اگر به كار نبريم حافظة مورد نظر از كلاس خودكار خواهد بود.
به طوري كه گفتيم، مي‌توان اعلان متغيرهايي از نوع اجتماع را به دنبال تعريف اجتماع به كار برد. پس مي‌توان صورت بالا را چنين تعريف كرد.

```
storage_class union tag {
        member 1 ;
        member 2 ;
        ...
        ...
        ...
        member m ;
} variable 1 , variable 2 , ... , variable n ;
```

ϖ **مثال 19.9** به اعلان زير توجه كنيد.

```
union id {
        char color [12] ;
        int size ;
} shirt , blouse ;
```

در اين مثال، دو متغير shirt و blouse از نوع اجتماع‌اند كه با نام id تعريف شده است. هركدام از اين دو متغير در هر لحظه يا معرف رشتۀ 12 كاراكتري color يا معرف مقدار صحيح size خواهد بود. بديهي است چون رشتۀ مزبور 12 كاراكتري است و به حافظۀ بيشتري در مقايسه با size نياز دارد، كامپايلر، به طور خودكار، 12 بايت حافظه به هر متغير اجتماع اختصاص خواهد داد.
ϖ
اجتماع ممكن است عضوی از ساختار باشد. همچنين ساختار ممكن است عضوی از اجتماع باشد.
ϖ **مثال 20.9** تعريف زير را در نظر بگيريد.

```
union id {
        char color[12] ;
        int size ;
        struct clothes {
        char manufacturer[20] ;
        float cost ;
        union id description ;
        } shirt , blouse ;
}
```

در اين مثال دو متغير shirt و blouse متغيرهايي از نوع ساختار ند كه با نام clothes تعريف شده‌اند و هركدام از آنها شامل اعضاي زير است: رشته‌ای به نام manufacturer (نام توليد كنندۀ لباس)، مقدار ی از نوع اعشار به نام cost (معرف بها ي لباس) و اجتماعی به نام description (نوع لباس). در اينجا ممكن است كه اجتماع معرف رشته يا color يا معرف مقدار صحيح يا size باشد. راه ديگري براي مشخص ساختن (اعلان) متغيرهاي ساختار shirt و blouse آن است كه دو نوع اعلان بالا را با يكديگر تركيب كنيم و به‌صورت فشرده‌تر زير بنويسيم.

```
struct clothes {
        char manufacturer[20] ;
        float cost ;
        union {
         char color[12] ;
         int size ;
        } description ;
} shirt , blouse ;
```

ϖ
نحوۀ دستيابي به عناصر يا اعضاي اجتماع مشابه همان است كه در مورد ساختارها گفتيم؛ يعني اين كار با استفاده از دو عملگر '.` و-`> انجام مي گيرد. بنابراين اگر variable متغير اجتماع باشد دستور variable. member يك عضو آن را معرفي و مشخص مي كند. به طريق مشابه، اگر ptvar متغير اشاره‌گر باشد كه به اجتماع اشاره مي‌كند، (يعني آدرس متغير ی از نوع اجتماع را در خود دار د) در اين صورت ptvar-> member به يك عضو آن اجتماع اشاره خواهد كرد.
ϖ **مثال 21.9** برنامۀ سادۀ زير را درنظر بگيريد.

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

```
union id {
            char color ;
            int size ;
          } ;
 struct {
            char manufacturer[20] ;
            float cost ;
            union id description ;
        } shirt , blouse ;
 printf ("%d\n" , sizeof (union id)) ;
 shirt. description. color = `w' ;    /* assign a value to color */
 printf ("%c %d \n" , shirt. description. color , shirt. description. size) ;
 shirt. description. size = 12 ;       /* assign a value to size */
 printf ("%c %d\n" , shirt. description. color , shirt. description. size) ;
}
```

در اين مثال، اولين عضو اجتماع تك‌كاراكتري است، برخلاف مثال قبلي كه آرايه اى 12 كاراكتري بود. انجام اين تغييرات بدين لحاظ صورت گرفته كه نسبت دادن مقادير به اعضاي اجتماع آسان‌تر باشد.

ملاحظه مي‌كنيد كه به عضو shirt. description. Color مقدار 'w`نسبت داده شده است . پس بايد توجه داشته باشيد كه عضو ديگر اجتماع، يعني shirt. Description مقدار معني نخواهد داشت. سپس با دستور printf، مقدار هر دو عضو اجتماع نمايش داده شده است. سپس به عضو shirt. description. size مقدار 12 نسبت داده شده است. بديهي است كه اين مقدار، روي مقدار حافظة مشترك كه براي دو عضو مزبور پيش‌بيني شده است خواهد نشست. سپس بار ديگر با دستور printf مقدار هر عضو نمايش داده شده است.

خروجي برنامة مزبور به شكل زير است.

```
2
w-24713
@ 12
```

سطر اول نشان مي‌دهد كه 2 بايت حافظه به اجتماع اختصاص داده شده است تا بتواند بزرگ ترين عضو خود را كه مقدار صحيح است در خود جاي دهد. در سطر دوم، داده اولي 'w` است كه معني و مفهوم دارد. اما داده دوم، يعني 24713-، معني و مفهومی ندارد. در سطر سوم، داده اولي @ است كه بدون معني است. اما داده دوم كه 12 است معني و مفهوم دارد.

شمارشي[1]

يكي از انواع داده‌هاي اسكالر نوع شمارشي است. بعضي زبانهاي ديگر مانند زبان پاسكال نيز اين نوع داده ها را در ميان انواع داده‌هاي استاندارد پشتيباني مي‌كنند. داده‌اى از نوع شمارشي، مشابه ساختار يا اجتماع است و اعضاي آن ثابتهايي اند كه به عنوان شناسه نوشته مي شوند، اگرچه مقدار يا ارزش آنها از نوع مقدار صحيح علامت دار است. اين ثابتها مقاديري را معرفي مي كنند كه مي‌توان به متغيرهاي شمارشي متناظر با آنها نسبت داد.

در حالت كلي نوع شمارشي به صورت زير تعريف مي‌شود.

```
enum tag {member 1 , member 2 , … , member m} ;
```

كه در آن enum كلمه‌اى كليدي است و tag نيز اسمي است كه داده شمارشي را مشخص مي‌كند و داراي اين تركيب است و عناصر member 1 , member 2 , … , member m نيز شناسه‌هايي را مشخص مي كنند كه ممكن است به متغيرهايي از نوع tag نسبت داده شود. اسامي اعضا بايد متفاوت و متمايز از يكديگر باشند.

وقتي كه نوع شمارشي تعريف شد، مي‌توان متغيرهاي شمارشي متناظر با آن را به صورت زير اعلان كرد.

```
storage_class enum tag variable1 , variable 2 , … , variable n ;
```

كه در آن storage_class گزينة اختياري است و كلاس حافظه را مشخص مي كند. enum نيز كلمه‌اى كليدي است كه بايد به كار برده شود. tag اسمي است كه در تعريف نوع شمارشي ظاهر مي‌گردد و بالاخره variable1 , variable 2 , … , variable n متغيرهاي شمارشي از نوع tag اند.

تعريف شمارشي را مي‌توان با اعلان متغيرها تركيب كرد و به صورت زير به كار برد.

```
storage_class enum tag {member 1 , member 2 , … , member m}variable 1 , variable 2 , … , variable n ;
```

در اين حالت، انتخاب نام براي مراجعه به آن اختياري است و مي‌توان آن را به كار نبرد. صورت بالا را مي‌توان به اختصار چنين نوشت.

```
enum enum_type_name {enumeration list} variable_list ;
```

كه در صورت مزبور به كار بردن نام نوع شمارشي، يعني enum_type_name، اختياري است. نام نوع شمارشي براي اعلان متغيرهايي از آن نوع است.

ϖ **مثال 22.9** قطعه برنامة زير نوع شمارشي‌اى به نام coin تعريف و نوع متغير money را از اين نوع اعلان مي‌كند.

```
enum coin {penny , nickel , dime , quarter , half_dollar , dollar} ;
enum coin money ;
```

با داشتن اين تعريف و اعلان، دستورهاي زير كاملاً درست و معتبر است.

```
money = dime ;
if (money = = quarter) prinf ("is a quarter") ;
```

ϖ

نكتة مهمي كه در اينجا بايد در مورد نوع شمارشي توجه داشت آن است كه هر سمبول معرف يك مقدار صحيح است و در هر عبارت از نوع مقادير صحيح به كار مى‌رود. براي مثال عبارت

```
printf ("the number of nickels in a quarter is %d" , quarter + 2) ;
```

كاملاً صحيح و معتبر است.

مقدار اولين سمبول شمارشي برابر صفر، مقدار دومين سمبول شمارشي برابر 1، و بالاخره مقدار n اُمين سمبول شمارشي برابر n – 1 است، مگر اينكه به طريق ديگري مقداردهي اوليه شده باشند . بنابراين عبارت printf ("%d %d" , penny , dime) ; مقادير 0 2 را در صفحة

[1]. enum

تصویر نمایش خواهد داد.
همچنین مي‌توان به هريك از سمبولها مقدار اوليه نسبت داد. براي اين كار بايد پس از سمبول مورد نظر علامت '=` و سپس مقدار صحيح مطلوب را به كار ببريم. وقتي كه به يكي از سمبول‌ها به طريق مزبور مقدار اوليه نسبت داده شد، سمبول بعدي مقدار بعدي را خواهد داشت ؛ يعني يك واحد از آن بزرگ‌تر خواهد بود. براي مثال دستور زير مقدار 100 را به quarter نسبت مي‌دهد.

```
enum coin {penny , nickel , dime , quarter = 100 , half_dollar , dollar} ;
```

و اكنون مقادير سمبولها به صورت زير خواهند بود.

```
penny    0
nickel   1
dime     2
quarter  100
half_dollar        101
dollar   102
```

به طور متعارف اين طور فرض مي‌شود كه سمبولهاي يك نوع شمارشي مي‌توانند به طور مستقيم ورودي يا خروجي باشند. ولي اين طور نيست. براي مثال قطعه برنامهٔ زير آنچه را انتظار داريم انجام نمي‌دهد.

```
money = dollar ;
printf ("%d" , money) ;
```

به خاطر داشته باشيد سمبول dollar به‌طور ساده مقدار رشته نيست ، بلكه اسمی براي مقدار صحيح است . بنابراين تابع printf نمي‌تواند رشتهٔ "dollar" را نمايش دهد. به طريق مشابه نمي‌توانيد با به كار بردن رشتهٔ همارز، به متغير شمارشي مقدار نسبت بدهيد . برای مثال دستور زير درست كار نمي‌كند.

```
money = "penny" ;
```

به هرحال به طريق ديگري مي‌توان مقدار رشته‌اي سمبولهاي مورد نظر را ايجاد كرد.

ϖ **مثال 23.9** قطعه برنامهٔ زير نوع سكه‌هايي را كه متغير money شامل است نمايش خواهد داد.

```
switch money {
        case penny:  printf ("penny") ;
                break ;
        case nickel:  printf ("nickel") ;
                break ;
        case dime:  printf ("dime") ;
                break ;
        case quarter:  printf ("quarter") ;
                break ;
        case half_dollar:  printf ("half_dollar") ;
                break ;
        case dollar:  printf ("dollar") ;
        }
```

ϖ

همچنين ممكن است آرايه‌اي از رشته‌ها تعريف كرد و مقدار متغير شمارشي را شاخص يا index آن آرايه به كار برد تا مقدار شمارشي را به رشتهٔ متناظر آن ترجمه كند.

ϖ **مثال 24.9** قطعه برنامهٔ زير رشتهٔ مورد نظر را به عنوان خروجي توليد خواهد كرد.

```
char name[ ] = {
         "penny" ,
         "nickel" ,
         "dime" ,
         "quarter" ,
         "half_dollar" ,
         "dollar"
        } ;
        printf ("%c" , name [(int) money]) ;
```

البته اين روش در صورتي درست كار مي كند كه به هيچ يك از سمبولها مقدار اوليه نسبت داده نشده باشد ، زيرا شاخص آرايهٔ رشته‌ها هميشه از صفر شروع مي‌شود.

ϖ

ϖ **مثال 25.9** فرض كنيد كه برنامهٔ C شامل دستورهای زير باشد.

```
enum colors { black , blue , cyan , green , magenta , red , white , yellow} ;
colors foreground , background ;
```

در سطر اول، نوع شمارشي به نام colors تعريف شده كه شامل هشت ثابت است و اسامي آنها عبارت‌اند از black , blue , cyan , green , magenta , red , white , yellow. در سطر دوم متغيرهاي foreground و background به عنوان شمارشي، از نوع colors اعلان شده‌اند. بنابراين به هريك از اين متغيرها مي‌توان هر يك از هشت ثابت black , blue , cyan ,... , yellow را نسبت داد.
مي‌توان دو دستور بالا را با يكديگر تركيب كرد و به صورت زير نوشت.

```
enum colors {black , blue , cyan , green , magenta , red , white , yellow}foreground , background ;
```

در اين مثال مقادير صحيح هشت ثابت بالا به صورت زير خواهد بود.

```
black                0
blue                 1
cyan                 2
green                3
```

magenta 4
red 5
white 6
yellow 7

به هرحال به طوري كه گفتيم، اگر به برخي از ثابتها مقدار اوليه نسبت داده شود، مقادير مزبور تغيير خواهد يافت . براى مثال اگر نوع شمارشي مزبور را به صورت

enum colors {black= -1 , blue , cyan , green , magenta , red = 2 , white , yellow} ;

تعريف كنيم، هشت ثابت مورد نظر مقادير زير را خواهند داشت.

black -1
blue 0
cyan 1
green 2
magenta 3
red 2
white 3
yellow 4

متغيرهاي شمارشي نمي‌توانند به طور كامل مشابه متغيرهاي از نوع مقدار صحيح م ورد پردازش قرار گيرند . مثلاً نمي‌توان به آنها مقدار جديد نسبت داد يا آنها را مقايسه كرد. همين طور نمي‌توان آنها را به عنوان ورودي به درون كامپيوتر خواند و به متغير شمارشي ديگر نسبت داد . اما مي‌توان مقدار صحيح را از ورودي دريافت كرد و آن را به متغير شمارشي نسبت داد. گرچه اين شيوه متداول نيست. همچنين فقط مي‌توان مقدار صحيح متغير شمارشي را به عنوان خروجي كامپيوتر نوشت.

حال دوباره دو دستور مذكور در آغاز اين مثال را درنظر بگيريد . با درنظر گرفتن دو دستور مزبور مي توان دستورهاي زير را به عنوان نمونه‌هايي از عمليات روي متغيرهاي شمارشي به كار برد.

```
foreground = white ;
background = blue ;
if (background = = blue)
   foreground = yellow ;
else
   foreground = white ;
if (foreground = = background)
    foreground = (enum colors) (+ + background % 8) ;
switch (background)
   {
      case black:  foreground = white ;
      break ;
      case blue:
      case cyan:
      case green:
      case magenta:
      case red:  foreground = yellow ;
            break ;
      case white:  foreground = black ;
            break ;
      case yellow:  foreground = blue ;
            break ;
      case default:  printf ("Error in selection of background color ") ;
 }
ϖ
```

خودآزمایی 9

1. ساختاري براي مشخصات دانشجويي زير بنويسيد.

شمارة دانشجو	نام	تاریخ تولد		
		ماه	روز	سال
123456	Amiri	11	5	1980

2. با استفاده از ساختار برنامه‌اي بنويسيد كه نام n نفر همراه با شماره تلفنشان را ذخيره كند. سپس شمار تلفن نام فردي را كه گرفته نمايش دهيد.
3. خروجي اين برنامه چيست ؟

```
union un{
   int i ;
   char c ;
} s ;
```

```
main ()
 {
   union un *p ;
   s.c = 'A' ;
   p = &s ;
   printf (" %d %c %d %c" , s.i , s.c , p-> i , p-> c) ;
}
```

4. برنامه‌اي بنويسيد كه با استفاده از اجتماعها توان اعشاري اعداد ($y = x^n$) را محاسبه و چاپ كند.


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)