# فصل پنجم
# زیربرنامه ها و DataType های کاربر

**by: Mahdi Sadeghizade**
**Email: Mahdi_sadeghe@yahoo.com**
**Site: www.Sadeghizadeh.ir**

## Abstract Data Type : ADT

یک ADT ، شامل یک DT به همراه مجموعه ای از عملیات روی آن می باشد . ( بدون در نظر گرفتن زبان برنامه سازی خاص)

مثال : در برنامه ثبت نام دانشجویان ، یک DT به نام Section تعریف می شود که شامل: نام درس ،نام استاد ، روز و ساعت تشکیل کلاس و عملیات روی شامل : ثبت نام دانشجو در Section و حذف دانشجو می باشد .

یک زبان برنامه سازی باید امکاناتی جهت ساختن و کار با ADT فراهم کند .

## جنبه های مجرد سازی ، تجرید یا انتزاع در زبانهای برنامه سازی

1. پایین ترین سطح انتزاع امکان تعریف زیر برنامه است .
2. سطح بعدی انتزاع امکان تعریف نوع داده جدید می باشد .
3. بالا ترین سطح انتزاع امکان تعریف نوع داده مجرد (ADT) می باشد . مثل Class,Package,…

## Abstraction

به معنی این است که پیاده سازی را از تعریف جدا کنیم .

یک زبان برنامه سازی باید ایده Abstraction را پشتیبانی نماید ؛ به این معنی که در ابتدا بیان کنیم که چه عملی باید انجام شود و سپس در سطح بعدی چگونگی انجام شدن آن را معین نماییم .

در واقع یک زبان برنامه سازی باید این امکان را فراهم نماید که طراحی نرم افزار به صورت لایه به لایه انجام شود که لایه های مختلف در کامپیوتر مجازی یک ایده ساده از طراحی لایه به لایه (یا تجرید) می باشد.

## پنهان سازی اطلاعات

در صورتی که یک زیر برنامه از زیر برنامه دیگری استفاده کند و به جزئیات زیر برنامه استفاده شونده دستیابی نداشته باشد ، ایده پنهان سازی اطلاعات پیاده سازی شده است .

مورد فوق برای استفاده از داده های نیز معتبر است . مثلا در مورد الگوریتم Sqrt (جذر) که جزئیات الگوریتم و نحوه نمایش اطلاعات از دید کاربر پنهان است .

پنهان سازی اطلاعات واژه ای است که به عنوان اصل مرکزی در طراحی مجردسازی تعریف شده توسط برنامه نویس استفاده می شود .

## فشرده سازی اطلاعات (Encapsulation)

در صورتی که بسته بندی اطلاعات یا همان فشرده سازی در Abstraction انجام شود ، نتایج زیر را خواهیم داشت :

- جزئیات داده های فشرده شده مخفی می ماند .
- اجازه دستیابی مستقیم به اطلاعات پنهان شده وجود ندارد.

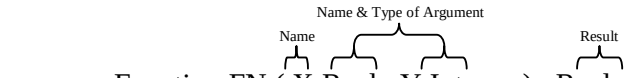
## ♦ زیر برنامه ها به عنوان عملیات مجردسازی

زیربرنامه های ابزاری برای پیاده سازی تجردسازی هستند بدین معنا که می توانند لایه های مختلف نرم افزاری را از هم جدا نمایند .

### مشخصات زیربرنامه ها :

1. نام زیربرنامه
2. تعداد ، نوع و ترتیب پارامترها
3. تعداد ، نوع و ترتیب نتایج
4. عملیات انجام شونده توسط زیربرنامه

مثال :




Function FN ( X:Real , Y:Integer ) : Real ;
FN : Real*Integer → Integer

### مشکلات زیربرنامه ها:

1. یک زیر برنامه ممکن است ورودی های ضمنی داشته باشد (متغیرهای سراسری)
2. یک زیر برنامه ممکن است به ازای ورودی های مشخصی تعریف نشده باشد (تقسیم بر صفر)
3. یک زیر برنامه ممکن است اثرات یا نتایج جانبی داشته باشد (تغییر پارامتهای ورودی)
4. یک زیر برنامه ممکن است خود اصلاح یا خودتغییر باشند یا حساسیت نسبت به قبل داشته باشند (متغیرهای Static در C )

### نکات مهم در پیاده سازی یک زیربرنامه :

1. عملیات TypeChecking و TypeConvertion لازم است انجام شود .

در این مورد بررسی نوع می تواند ایستا باشد ، در صورت اعلان صریح داده ها و در غیر این صورت به صورت پویا در زمان اجرا انجام می شود .

2. یک زیربرنامه شامل دو قسمت : Local Data Variable و Body می باشد .

در برخی از زبانها مثل Pascal,Ada (نه در زبان C ) بدنه می تواند شامل تعریف زیربرنامه های دیگری باشد.

## تعریف کردن زیربرنامه و فعال سازی آن (اجرا)

تعریف زیربرنامه ، اطلاعات و مشخصات لازم را جهت فعال سازی آن فراهم می کند .

تعریف زیربرنامه ، مشابه تعریف یک Type و فراخوانی یا فعال سازی آن معادل تعریف متغیر می باشد .

LifeTime یک زیربرنامه از لحظه شروع فراخوانی (call) تا لحظه خاتمه یافتن زیربرنامه (return) می باشد .

**اطلاعاتی که تعریف زیربرنامه جهت فعال سازی آن فراهم می کند شامل موارد زیر است :**

Data/Stack Segment ← Dynamic
1. ذخیره پارامترها
2. ذخیره نتایج زیربرنامه
3. ذخیره متغیرهای محلی

Code Segment ← Static
4. ذخیره مقادیر ثابت و لیترالها
5. ذخیره کد تولید شده جهت اجرا

مثال

```
Function   FN   (x:real , y:integer) : real
           Const  max=20;
           Var
                  M:array[1..max]of  real;
                  N:integer;
            Begin
                  N:=max;
X:=2*x+m[5];
End.
```

**برای فعال سازی زیربرنامه دو قسمت Static و Dinamyc وجود دارد :**

- در قسمت static کد اجرایی و ثابت ها قرار دارند که در کدسگمنت ذخیره می شوند .
  این قسمت برای هر زیربرنامه یک بار ایجاد می شود هرچند زیربرنامه چندین بار فراخوانی شود .
- **Prologue** : یک قسمت از کد است که برای عملیات انتقال پارامترها ، push کردن فلاگ ها و رجیسترها از آن استفاده می شود و به طور کلی شامل عملیاتی است که در شروع فراخوانی باید انجام شود .(عملیات یا دنباله فراخوانی)

دستورات به عنوان کد اجرایی : تمامی دستورات اجرایی و تولید شده در بدنه اصلی زیربرنامه در این قسمت قرار می گیرند .

- **Epilogue** : شامل عملیاتی است که برای انتقال نتایج ،pop کردن فلاگ ها ،رجیسترها و مانند آن انجام می شود و به طور کلی شامل عملیاتی است که در پایان فراخوانی زیربرنامه باید انجام شود .( عملیات ارجاع یا دنباله بازگشت)

- Ar (Activation Record) : Dynamic

این قسمت به ازای هر بار فراخوانی برنامه ایجاد خواهد شد و هر بار که فراخوانی تمام شود از بین می رود.

☑ اندازه و ساختار ARمورد نیاز برای زیربرنامه می تواند به طور معمولی و در ضمن ترجمه (Compile ) مشخص شود مگر این که داده هاییی با طول متغییر داشته باشیم ( آرایه نامحدود)

شکل زیر نحوه استفاده از قسمت Static و Dynamic را برای تابع FN نشان می دهد .

### زیربرنامه های چند هدفه

منظور این است که چندین زیربرنامه با اسامی یکسان وحود دارد که تعداد پارامترها،نوع و ترتیب آنها متفاوت است .

ازجهت پیاده سازی کامپایلر با توجه به تعداد پارامترها ، نوع و ترتیب آنها تشخیص می دهد که کدامیک از زیربرنامه ها را فراخوانی کند و کد لازم برای فراخوانی زیربرنامه مربوطه را تولید کند .

## زیربرنامه ها به عنوان DO

نکات مهم در این باره شامل موارد زیر است :

- در بعضی از زبانها مثل Pascal , fortran , Java , C , C++ کامپایلر با توجه به تعریف زیربرنامه فرم اجرا را تولید خواهد کرد و در طی اجرای زیربرنامه دیگر قابل دسترس نیست و نم توان تعریف آن را تغییر داد . (ایستا)
- در بعضی از زبانها مثل Lisp , Snobol , Prolog , Perl (زبانهای مفسر نرم افزاری) امکاناتی جهت تعریف زیربرنامه و تغییر آن در حین اجرا وجود دارد (پویا)

☑ در مورد اول انعطاف پذیری کم ولی سرعت اجرا زیاد می باشد در حالتی که در مورد دوم عکس آن می باشد.

## تعریف نوع

یک زبان برنامه سازی باید امکاناتی جهت تعریف نوع جدید با توجه به نوع های موجود در زبان را فراهم کند . هر نوع شامل یک نام و یک توصیفگر می باشد .

مثال : Type RealVect=Array[1..10]of Real

☑ تعریف نوع به عنوان الگویی برای ساخت اشیاء داده در طول اجرای برنامه استفاده می شود .

```
Struct Rtype
{
Int num;
Int den;
}
Struct Rtype A,B,C;
```

این مطلب که کلمه کلیدی struct قبل از انواع داده جدید در هنگام تعریف متغیر می بایست ظاهر شود ، یکی از اصول مجرد سازی داده ها را نقض می کند . تمایل در این است که نوع داده جدید به عنوان توسعه ای برای زبان ظاهر شود و از نظر نحوی و معنایی مشابه انواع داده اصلی می باشد .
این مشکل در زبان C به علت طول عمر زبان است که بعدا از کلمه کلیدی Typedef برای رفع آن استفاده شد . در زبان C انواع داده جدید توسط کلمه struct ایجاد می شود در حالی که typedef نوع جدیدی را ایجاد نمی کند و فقط جایگزین با تعریفش می باشد .

```
Struct Rtype
{
Int num;
Int den;
}
Typedef struct rtype
Struct Rtype A,B,C;
```

**مزایای تعریف نوع :**

1. ساده کردن ساختار برنامه
2. جلوگیری از تکرار نوع های موجود در زبان
3. ساده کردن انتقال پارامترها
4. یک شکل جدید از بسته بندی و پنهان سازی اطلاعات را فراهم می کند .

☑ از جهت پیاده سازی کامپایلر باید هر نوع جدید را با توجه به نام و توصیفش در جدولی وارد نماید و به هنگام اجرای کد و انتقال پارامترها و عملیات دیگر از آن کد استفاده کند .

**معادل بودن نوع ها**

سئوالی که در اینجا مطرح است ، این است که دو نوع در چه صورتی با هم معادلند؟

```
Type VECT1 Array[1..10]of Real ;
     VECT2 Array[1..10] of Real ;
Var   x,z :VECT1
        y:VECT2;
Procedure sub(A:VECT1);
       Begin
```

```
        End;
Begin
        x:=y;
        sub(y);
end.
```

## انواع معادل سازی

1. معادل سازی نام (Name Equivalence) :

اگر یک زبان برنامه سازی معادل سازی نام داشته باشد دو نوع در صورتی یکسان هستند که نام آنها یکسان باشد .

در مثال قبل هر دو دستور نادرست هستند ، زیرا متغیرهای x و y از دو نوع متفاوت خواهند بود چون نامشان یک نیست .

در زبانهای Ada (تحقیقی) ، C++ و Pascal (پارامترهای زیربرنامه) از این نوع معادل سازی استفاده می شود .

**نقاط ضعف :**

- هر نوع باید یک نام داشته باشد و نوع های بدون نام را نمی توان داشت ،در این حالت انتقال پارامترها دچار مشکل می شود.

مثال:
```
Var w:Array[1..10]of Real;
```
در این مثال نمی توان w را به زیر برنامه sub انتقال داد .

- یک Single type definition باید در کل برنامه و به صورت Global حاکم باشد .

تعاریف کلاس در زبان C++ و package در زبان Ada و فایل های header در زبان C مطمئن می سازند که چنین شرایطی برقرار است .

### 2. معادل سازی ساختار (Structural Equivalence)

دو نوع معادل هستند اگر ساختار آنها و جزئیات تمام مؤلفه ها یکسان باشد . در این حالت انعطاف پذیری زبان برنامه سازی بالاست (سرعت آن پایین است)زبانهای قدیمی مثل C,fortran,cobol,PL1 دارای تعاریف نوع نمی باشند و بنابراین یک شکل خاص از معادل بودن ساختار را استفاده می کنند.

**نقاط ضعف:**

اگر نام مؤلفه ها یکسان و ترتیب آنها متفاوت باشد و یا نام مؤلفه ها متفاوت و ترتیب آنها یکسان باشد چه تاثیری در معادل بودن نوع دارد؟

جهت تشخیص معادل بودن نوع باید هزینه پرداخت شود ، یعنی زمان را از دست می دهیم . در حقیقت کامپایلر باید زمان صرف کند که آیا دو نوع معادلند یا خیر .
دو نوع به صورت ناخواسته ممکن است یکسان باشند در حالی که از دید برنامه نویس متفاوتند و در عین حال زبان برنامه سازی کمکی به static type checking نمی کند .
مثال:

```
Type METERS=integer;
LITERS=integer;
Var LEN:METERS;
VOL=LITERS;

X:=LEN+VOL;
```

زبان پاسکال به طور کامل از هریک از این روش ها استفاده نمی کند . در زیربرنامه ها بیشتر به صورت نامی و در بقیه حالات بیشتر به صورت ساختاری عمل می کند .

## تعریف نوع پارامتریک

مثال

```
Type section(MAX-SIZE:integer)IS RECORD
      Room : integer
      Instructor : integer;
      Class-Size : integer;
      Class-Roll : Array[1..MAX-SIZE]
End Record;

x : section(10);
y : section(23);
```

☑ در زبان ML خود نوع را نیز می توان به عنوان پارامتر در نظر گرفت .

از جهت پیاده سازی کامپایلر باید ارزش پارامتر را محاسبه کرده و سپس با توجه به ارزش پارامتر و نوع مشخص شده ،نوع جدیدی از متغیر تعریف خواهد شد .
در واقع کامپایلر باید دو مر حله را طی نماید :

1. ارزش یابی پارامتر
2. تعریف نوع جدید

## تجرد سازی انواع اطلاعات (ADT)

ADT نوع جدیدی است که توسط برنامه نویس تعریف می شود و شامل موارد زیر است :

1. تعریف مجموعه ای از DO ها . معمولا از یک یا چند نوع متنوع(متغیرها و انواع آن معمولا در اول کلاس (ADT) تعریف می شوند)
2. تعریف مجموعه ای از عملیات روی DO ها (روالهای داخل کلاس)
3. بسته بندی کل مجموعه به گونه ای که دستیابی به DO ها تنها از طریق عملیات بند 2 باشد (از طریق زیرروالها)

- ☑ ADT در Ada توسط دستور package پشتیبانی می شود و در زبانهای C++,smalltalk,Java توسط دستور class پیاده سازی می شود .
- ☑ انواع داده مجرد ایجاد شده توسط برنامه نویس اغلب به عنوان کتابخانه ای خاص از زیر برنامه ها ر زبان ظاهر می شوند.مثل زبانهای C,fortran,pascal .
- ☑ Ada و C++ در این بین معدود زبانهایی هستند که به طور گسترده با جنبه های مجرد سازی داده ها به کار می روند .