فصل ۱۱

اشاره گرها چیستند؟

اشاره گر را میتوان به معنای واقعی یک مفهوم اساسی در برنامه نویسی دانست، درک آن مشکل است ولی زمانی که آن را بشناسیم واقعا بهتر برنامه مینویسیم، شاید چون در خیلی از موارد عملکردها را حس میکنیم!

اشاره گرها یک نوع داده هستند مثل Char،Integer و همانند اینها که در واقع آدرسهای حافظه را ذخیره میکنند. مساله از این جهت مهم است که هر چیزی که استفاده میکنیم حتما یک محل دارد و بنابراین راه دسترسی به آن همین اشاره گرها هستند. ساده؛ اشاره گر به یک رشته (String) اشاره میکند، وقتی از یک شئ (Object) استفاده میکنیم یا حتی زمانی که یک رویداد (Event) را صدا میزنیم، همه و همه اشاره ای به یک اشاره گر هستند. فرض کنید میخواهیم در صفحه نمایش ایجاد افکتهای گرافیکی کنیم، این کار از طریق نوشتن مقادیر مورد نظر در نقاط صفحه نمایش (Pixels) امکان پذیر است، پس کافی است اشاره گری به اولین نقطه داشته باشیم و کل نقاط در اختیار ما هستند.

البته اشاره گرها در دلفی آنگونه که در زبانهای C و C++ مورد توجه هستند، استفاده نمیشود، خصوصا به این دلیل که در Object Pascal سعی شده تا حد امکان از درگیری مستقیم برنامه نویس با اشاره گرها جلوگیری شود.

چرا به اشاره گرها نیاز داریم؟

خوب! واقعا من به عنوان یک برنامه نویس چه نیازی به دانستن محل حافظه یک متغییر دارم!؟

یک جا نوشته بود: "اشاره گرها میتوانند به هر چیزی اشاره کنند، پس به آنها نیاز داریم شما میتوانید با اشاره گری به یک شئ اشاره کنید و آنرا تغییر دهید. بعد به راحتی با تغییر آدرس اشاره گر مربوطه به یک شئ دیگر اشاره میکنید و آنرا تغییر میدهید هر چیزی –و به معنای واقعی هر چیزی- که در یک برنامه استفاده میشود در محلی از حافظه قرار گرفته است.

جالب است اگر به این صورت به اشاره گرها نگاه کنیم که آدرس داده های برنامه را در زمان اجرا در اختیار ما قرار میدهند، حتی آنهایی که در زمان اجرا ایجاد شده اند و شاید حتی نام مشخصی هم نداشته باشند.

استفاده از اشاره گرها در دلفی

گفته شد اشاره گرها هم یک نوع داده هستند، پس باید امکان تعریف کردن آنها را داشته باشیم، اما چگونه؟

برای تعریف اشاره گرها از علامت ^ استفاده میشود. مثال زیر تعریف اشاره گری به یک داده از نوع Integer را نمایش میدهد:

```
Var
```

```
 pntMyInt: ^Integer;
```

اما بعد از تعریف اشاره گر چگونه یک آدرس به آن نسبت دهیم؟ گفته شد اشاره گرها میتوانند به هر چیزی اشاره کنند، یک متغییر، یک تابع یا هر چیز دیگری، برای نسبت دادن هر کدام از اینها به یک اشاره گر از علامت @ استفاده میشود:

```
Var
 tmpInt: Integer;
 pntMyInt: ^Integer;
Begin
…
pntMyInt := @tmpInt;
…
End;
```

دقت شود کد فوق مقدار متغییر tmpInt را به pntMyInt منتقل نمیکند، بلکه فقط آدرس محل حافظه ای که tmpInt در آن قرار دارد را در pntMyInt ذخیره میکند. فرض کنید بخواهیم مقدار ذخیره شده در tmpInt را تغییر دهیم. کد زیر را ببینید:

```
Var
 tmpInt: Integer;
 pntMyInt: ^Integer;
Begin
…
tmpInt := 100;
…
pntMyInt := @tmpInt;
…
pntMyInt^ := 102;
…
End;
```

کد فوق را بررسی میکنیم؛ در ابتدای کار با استفاده از روش مقدار دهی مستقیم عدد ۱۰۰ را به متغییر tmpInt نسبت دادیم. سپس در مراحل بعدی آدرس محل حافظه این متغییر را در اشاره گر pntMyInt ذخیره کردیم، توجه توجه! این کار با مساوی قرار دادن دو متغییر کاملا متفاوت است و در اینجا عدد ۱۰۰ در اشاره گر ما قرار نمیگیرد.

اما pntMyInt^ := 102 به چه معنی است؟ خیلی راحت: عدد ۱۰۲ را در محل حافظه ای که pntMyInt به آن اشاره دارد ذخیره کن، و آن محل حافظه جایی نیست جز متغییر tmpInt. چه اتفاقی افتاد؟ tmpInt := 102 ، به همین راحتی! هر گاه علامت ^ بعد از اشاره گر قرار گرفت مقدار ذخیره شده در آن محل حافظه مد نظر است.

خوب! پس ما میتوانیم محل حافظه ای که اشاره گر به آن اشاره میکند را بخوانیم یا بنویسیم (تغییر دهیم)، همچنین مقدار متغیری که در محل حافظه ای که اشاره گر به آن اشاره دارد را بخوانیم یا بنویسیم.

گفته شد که اشاره گرها به اشیاء هم اشاره میکنند پس نباید عجیب باشد اگر جایی ببینیم:

```
var;Number : ^Integer;
Name : ^String;
formPointer : ^TForm1;
```

اجازه دهید کمی درگیرتر شویم! با یک کد شروع میکنیم:

```
type
 TMyObject = record
   MyCode: Integer;
   MyText: String;
 end;

 pntMyObject = ^TMyObject;
...
procedure CheckMyObjectPointer;
var
  Object1, Object2: TMyObject;
  Pointer2Object: pntMyObject;
begin
  Pointer2Object := @Object1;
  Pointer2Object^.MyCode := 25;
  Pointer2Object^.MyText := 'First;'
```

```
  Pointer2Object := @Object2;
  Pointer2Object^.MyText := 'Second;'
end;
```

به نظر پیچیده هست ولی اگر کمی دقت کنیم خیلی هم مشکل نیست. بیایید با هم این کد را بررسی کنیم:

در قسمت تعریف داده ها یک نوع شئ شخصی تعریف شد، خیلی هم مهم نیست، شما میتوانید از اشیاء موجود دلفی استفاده کنید. اما در قسمت دوم تعریف انواع داده ها هم اتفاق خاصی نیافتاده است. این قسمت را اینجا آوردم تا با این حالت هم آشنا شوید. شما میتوانید انواع اشاره گرها را به این صورت استفاده کنید، در واقع در اینجا هر وقت در برنامه بخواهیم اشاره گری به TMyObject تعریف کنید به جای ^TMyObject به راحتی از pntMyObject استفاده میکنید. اما در رویه (Procedure) نوشته شده، ابتدا در بخش تعریف متغیرها دو شئ از نوع TMyObject تعریف میشود ونیز یک اشاره گر به همان نوع شئ. در ابتدای رویه اشاره گر با آدرس شئ اول تنظیم میشود و خیلی راحت بااستفاده از مواردی که در بالا گفته شد مقدار دهی میشود. سپس اشاره گر ما به شئ دوم نسبت داده میشود و آدرس محل حافظه شئ دوم در آن قرار میگیرد

خواص (Property) یک شئ را در نظر بگیرید که به این صورت قابل تغییر هستند!

شاید از خودتان بپرسید چرا من مسیر خود را طولانی کنم؟ گفته بودم که دردلفی نیاز به اشاره گرها مانند C و C++ احساس نمیشود، اما به شما میگویم گاهی این کار مسیر شما را خیلی بیشتر از آن چیزی که فکر میکنید کوتاه میکند، استفاده از بعضی توابع API را در نظر بگیرید...

اشاره گرهای عمومی در دلفی

تا اینجا از اشاره گرهایی استفاده کردیم که به نوع داده مشخصی اشاره داشتند، در این حالت دلفی به اندازه کافی باهوش هست که اشتباهات ما را در استفاده از اشاره گر به ما گوشزد کند. اما گاهی مجبوریم از اشاره گرهای بی نام و نشانی استفاده کنیم که به آنها اشاره گر عمومی میگوییم و قادرند به هر چیزی بدون در نظر گرفتن نوع آن اشاره کنند. این نوع اشاره گرها در دلفی به نوع داده ای Pointer اطلاق میشوند. به مثال زیر توجه کنید:

```
Var
Something : Pointer;


procedure CheckGenericPointer( const pntGPointer: Pointer);
var
 tmpInt: ^Integer;
 MyCode: Integer;
begin
 tmpInt := pntGPointer;
```

```
 MyCode := tmpInt^;
...
end;
procedure CallCheckProc;
var
 MyInt: Integer;
begin
 MyInt := 100;
 CheckGenericPointer( @MyInt);
end;
```

در کد فوق دو مورد در نظر گرفته شده است، هم از اشاره گرهای عمومی استفاده شده و هم نشان داده شده است که چگونه میتوانیم از اشاره گرها به عنوان پارامتر توابع استفاده کنیم. نکته ای که باید در نظر بگیریم نحوه استفاده از اشاره گر عمومی است. یک اشاره گر عمومی ابتدا باید به یک اشاره گر دیگر نسبت داده شود تا قابل استفاده باشد. باقی موارد پیش از این بررسی شده اند. قبول دارم که استفاده از اشاره گرها در این کد منطقی نیست، پس کجا از اشاره گرهای عمومی استفاده کنیم؟ تا به حال از آرایه های پویا استفاده کرده اید؟ هر جایی که به نوعی با مفهوم حافظه پویا (Dynamic Memory) سر و کار داشته باشیم اشاره گرها به کمک ما می آیند، هر چند دلفی در اکثر این موارد نیز کار ما را آسان نموده است. برای نمونه TList را در نظر بگیرید، نگاهی به مثالهای همراه دلفی برای این کلاس بیاندازید، ضرر نمیکنید...

اشاره گرهای Nil

فکر نمیکنم کلمه مناسبی برای بیان این مفهوم پیدا شود. یک اشاره گر که به هیچ عنصری نسبت داده نشده باشد میتواند خطرناک باشد. اشاره گرها به ما اجازه میدهند مستقیما با حافظه کامپیوتر کار کنیم. خوب پس با اشاره گرهایی که –فعلا- به چیزی اشاره نمیکنند چه کنیم؟ راه حل این مشکل nil است. nil ثابتی (Constant) است که میتوانیم آنرا به هر اشاره گری نسبت دهیم. وقتی nil به اشاره گری نسبت داده شود، آن اشاره گر به جایی رجوع نمیکند مثل یک آرایه پویای خالی! به این صورت از بروز حوادث نامطلوب پیشگیری میکنیم. کد زیر نمونه این کار را نمایش میدهد:

```
pntMyPointer := nil;
```

طبیعی است که شما قصد استفاده از این اشاره گر را دارید بنابراین قرار نیست همواره به این حالت باقی بماند، بنابراین نیاز به کنترل آن در قسمتهای دیگر برنامه دارید، کد زیر را ببینید:

If pntMyPointer <> nil then

اگر nil بود آنرا به محل مورد نظر خود نسبت دهید. راه دیگر استفاده از تابع Assigned است:

If Assigned(pntMyPointer) Then

این مورد نیز در راهنمای همراه دلفی با مثال بررسی شده است.

اشاره گر از نوع رکورد

همانطور که اشاره گرهایی از نوع ساده مثل مقادیر عددی-کاراکتر-رشته-اعشاری و... تعریف میکردیم میتوانیم اشاره گرهایی از نوع رکورد تعریف کنیم

```
Type
Rec=record
Id:integer;
Name:string[50];
End;
Prec:^rec;

Var
One,two:prec;
```

در مثال فوق رکوردی بنام rec تعریف شد و اشاره گری از نوع Prec که به rec اشاره میکند و ۲ متغییر one وtwo که متغییر هایی از نوع رکورد هستند برای دستیابی به اجرای رکورد با استفاده از اشاره گرهای رکورد به صورت زیر عمل میکنیم

```
One^.id:=10;
One^.name:=mori;
I:=two^.i;
St:=two^.name;
```

تخصیص خودکار حافظه

وقتی شما از نوع‌های پایه (Integer، real،word و...) برای ایجاد متغییرهای خود استفاده می کنید، هیچ نگرانی درباره تخصیص حافظه آن وجود ندارد چون دلفی

خودش آنرا تخصیص حافظه می کند و سپس آزاد میکند.

```
type TDay = (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)
var
Name : String; {256 Bytes}
 X, Y : Integer; {4 + 4 = 8 Bytes}
 List : array [0..10] of Double; {8 * 11 = 88 Bytes}
 Today : TDay; {1 Byte{
```

دراین نمونه پس از پایان برنامه، تمام حافظه تخصیص داده شده فراخوانی و آزاد می شود.

تخصیص بلاکی از حافظه

ما می‌توانیم از اشاره به بلاک‌هایی از تخصیص حافظه در سیستم استفاده کنیم. این کار را با رویه‌های GetMem و FreeMem برای تخصیص و آزاد سازی حافظه استفاده میکنیم.

```
;varSomething : PointerbeginGetMem(Something, 100);FreeMem(Something, 100);end
```

اشاره به حافظه از قبل تخصیص داده شده

هر دو نوع اشاره‌گرها می توانند به هر جایی از حافظه اشاره بکنند. این بدان معناست که آنها می‌توانند اشاره به فضای اشغال شده با داده‌هایی که در حال حاضر موجودند داشته باشند. این نمونه اشاره‌گر احتیاجی به تخصیص حافظه ندارد.

```
varSomething : Pointer;
MyString : PChar;
type PChar = ^Char;
begin
GetMem(Something, 100);
MyString := Something;
StrCopy(Something, 'Hello World');
FreeMem(Something, 100);
end;
```

حافظه Heap

Heap شامل قسمتی از حافظه موجود در یک برنامه است که آنرا حافظه پویا می نامیم. Heap بخشی است که در آن تخصیص و تعریف حافظه به صورت تصادفی (Random) اتفاق می‌افتد. این به آن معناست که اگر شما سه بلاک از حافظه را به طور متوالی تخصیص دهید، می توانید بعد از هر دستور آنرا از بین ببرید. مدیر Heap جزئیات را برای شما نگهداری می کند. بنابراین شما به سادگی

می توانید یک حافظه جدید را با GetMem و یا بوسیله صدا زدن constructor هنگام ساختن یک شی درخواست کنید و دلفی به شما یک بلاک جدید را برخواهد گرداند. Heap یکی از سه فضای موجود در برنامه کاربردی را استفاده کرده و دوتای دیگر به صورت فضای یکپارچه (Global) و پشته قرار می گیرند.

حافظه Stack

Stack شامل قسمتی از یک بخش از حافظه موجود یک برنامه است که دینامیکی است اما برای تخصیص و آزادسازی فرامین مخصوص دارد. تخصیص Stack به صورت LIFO می باشد. این بدان معناست که آخرین حافظه شیء شما تخصیص داه خواهد شد و سپس حذف می شود. حافظه پشته در روتین‌های نوعی استفاده می‌شود. وقتی شما یک روتین را صدا میزنید، پارامترهایش و روتین نوع آن در پشته ریخته می شود. همچنین پارامترهایی که در یک روتین تعریف میشوند، در پشته ذخیره میشوند و وقتی روتین خاتمه پیدا می کند تمام آنها به طور خودکار از بین می رود

تابعAdder:

این تابع ادرس یک متغییر یا یک روتین را برمیگرداند

```
function Addr(X): Pointer;
```

مثال:

```
var
 iValue    : integer;
 pIntValue : ^integer;
begin
  iValue := 2001;
  pIntValue := Addr(iValue);
```

تابعPtr:

یک ادرس را تبدیل به اشاره گر میکند

```
function Ptr(Address: Integer): Pointer;
```

تابع New

برای تخصیص حافظه به صورت پویا استفاده می شود

```
Var
Int:Pinteger;
Begin
New(int);
Int:=10;
```

میتوان به صورت دیگر هم نوشت

```
Type
Int=^integer;
Var
Int1:int;
Begin
New(Int1);
Int:=10;
```

تابع dispose

برای ازاد سازی حافظه تخصیص داده شده استفاده می شود

```
Var
Int:Pinteger;
Begin
New(Int);
Int:=10;
Showmessage(inttostr(Int));
Dispose(Int);
```

فصل ۱۲

زیر برنامه ها

در اين جلسه مي خواهيم به نحوه نوشتن زير برنامه هايي كه در يك برنامه نيازمند استفاده از آنها هستيمبرای نوشتن برنامه های بزرگ باید انرا به بخشهای کوچکتر تقسیم نمود یا در بیشتر برنامه ها نیازمند این هستیم که عمليات خاصي انجام شود و به طبع آن نتايجي حاصل آيد و اين عمليات در قسمت هاي مختلف برنامه چنديدن بار مورد استفاده قرار مي گيرند و اگر قرار باشد این کدها چند بار نوشته شود اول اینکه موجب اتلاف وقت شده و ثانیا از کارايي برنامه كاسته مي شود به همين خاطر با نوشتن اين كدها به عنوان تابع يا زيرروال مي توان به دفعات از اين كدها در هر جاي برنامه كه نياز باشد استفاده كرد . محل نوشته توابع وروال ها در کد برنامه قبل تمام رویداد ها و بعد از قسمت implementation مي باشد . زير برنامه ها به ۲ بخش تقسیم می شوند توابع Function و رویه Procedure که اینها نیز به ۲ بخش تقسیم می شوند زیر برنامه های با پارامتر و بدون پارامتر . پارامترها متغییر هایی می باشند که به زیر برنامه ها ارجا داده می شوند

مزایای ایتفاده از توابع رو رویه ها

خوانایی برنامه را بالا می برد

امکان برنامه نویسی گروهی را فراهم می اورد

از این زیر برنامه ها میتوان به دفعات در برنامه های مختلف استفاده نمود

از این زیر برنامه ها میتوان در زیر برنامه های دیگر استفاده نمود

سرعت برنامه نویسی به علت استفاده از زیر برنامه های از قبل اماده بالا می رود

خطایابی بسیار اسان می شود زیر عدم استفاده از زیر برنامه در برنامه باعث طولانی شدن کدها می شود و همچنین زیر برنامه هایی که در برنامه های دیگر استفاده شده اند از صحت عملکرد صحیح ان اطمینان داریم

-نوشتن زیرروال :

Procedureها يا رويه ها براي مواقعي به كار مي روند كه بخواهيم عملياتي انجام شوند و نتايجي حاصل شود .
فرم كلي نوشتن روال به صورت زير مي باشد :

```
Procedure نام روال(پارامترها);
متغیرها در صورت نیاز تعریف
begin
دستور و یا دستورات ;
end ;
```

-كلمه Procedure يك قسمت ثابت مي باشد. و بين كننده نوع زير برنامه است.
-قسمت پارامترها : در اين قسمت ليستي از پارامترهايي است كه در صورت نياز لازم است به روال داده شود تا بر اساس آن متغيرها عمليات خاصي انجام شود. و به انها متغییر های محلی گویند.کدها بین بلاک Begin و End قرار میگیرند
نحوه نوشتن پارامترها به صورت زير مي باشد :

```
(... ; نوع متغیر:نام متغیر)
```

نكات :
***پارامترها با كاركتر ; از هم جدا مي شوند و در صورتي كه چند پارامتر از يك نوع باشند مي توان با استفاده كاراكتر , از هم جدا و سپس نوع آنها را ذكر نمود . توجه داشته باشيد كه ترتيب پارامترها از چپ به راست است .
***اين نوع پارامترها را پارامترهاي ورودي مي گويند يعني تنها فقط براي ورود اطلاعات به روال مي باشد و هيچگونه تغييري در اصل آنها پيدا نمي كنند .

***پارامترهايي داريم به نام ورودي - خروجي يعني هم براي ورود اطلاعات به كار مي روند و هم براي اينكه از روال اطلاعاتي را دريافت كنيم ، براي مشخص كردن پارامتر ورودي - خروجي بايد قبل از نام متغير كلمه كليدي var را بنويسيم .
***يك روال مي تواند بدون پارامتر باشد .

سه مثال ساده :

رویه بدون پارامتر

```
Procedure sum ( ) ;
Var A,B,C : integer;
Begin
A := StrToInt ( InputBox ( ' Enter Number ' , ' Enter number 1 : ' , ' 0 ');
B := StrToInt ( InputBox ( ' Enter Number ' , ' Enter number 2 : ' , ' 0');
C := A + B;
ShowMessage ( IntToStr ( C ));
End;
```

رویه های دارای پارامتر

```
Procedure Msg(Ok : Boolean) ;
begin
if Ok then
ShowMessage('The parameter is true')
else
ShowMessage('The parameter is false') ;
end ;
---------------------------------
Procedure Set(Ok : Boolean ; var St : String);
begin
if Ok then
St := 'Ok'
else
St := 'Not Ok' ;
end ;
```

اين روال ۲ پارامتر مي گيرد ، پارامتر اول از نوع بولي مي باشد كه به عنوان ورودي عمل ميكند و پارامتر دوم كه از نوع رشته اي مي باشد به صورت ورودي - خروجي عمل مي كند . اين روال دو پارامتر را مي گيرد و بر طبق True و يا False بودن متغير Ok مقدار متغير رشته اي St را تنظيم مي كند .

نوشتن تابع :

*توابع براي موقعي به كار مي روند كه بخواهيم پس از انجام عملياتي خاص نتيجه و مقداري را براي ما بر گرداند .
فرم كلي نوشتن توابع به صورت زير مي باشد :

```
Function نام(پارامترها) : تابع نوع مقدار بازگشتي ;
تعريف متغيرها در صورت نياز
begin
دستور و يا دستورات ;
Result := مقدار برگشتي تابع
end ;
```

Function يک قسمت ثابت و بيان کننده نوع زير برنامه می باشد .توابع نيز همانند روالها ميتوانند با پارامتر يا بدون پارامتر باشند.همانطور که گفتيم توابع بايد يك مقدار را برگردانند از اين رو بايد نوع مقدار برگشتي مشخص شود براي اين کار نوع مقدار را جلوي نام تابع بعد از پرانتز پارامترها و علامت : مي نويسيم . در واقع با اين کار تابع را همانند متغير ها تعريف مي کنيم و نام تابع نيز همانند نام متغيرها عمل مي کند. براي اينکه به نتيجه تابع مقدار دهيم از دستور Result استفاده مي کنيم ، نوع مقداري که جلوي دستور Result نوشته مي شود بايد هم نوع مقدار برگشتي تابع باشد . نکاتي که در مورد روال ها گفتيم براي توابع نيز به کار ميروند .

دو مثال ساده :

```
Function zaman( ) : string;
Begin
Result := TimeToStr ( Now) ;
End ;
```

تابع بدون پارامتر

```
Function Equal(j : Integer) : Boolean ;
var
y : Integer ;
begin
y := 1383 ;
if j=y then
Result := True
else
Result := False ;
end ;
```

تابع دارای پارامتر

نحوه فراخواني و اجراي روال ها و توابع :

-براي اجراي توابع و روال هاي تنها كافي در جايي از برنامه كه نياز است نام تابع و يا روال مورد نظر را نوشته و در صورت داشتن پارامتر آنها را نيز مي نويسيم . به طور مثال :

روال:

```
Msg (True) ;
```

True مقداری از نوع Boolean می باشد که به روال Msgپاس داده شده

تابع :

```
var
g : Boolean ;
i:integer;
begin
i:=99
g := Equal(i) ;
label1.caption:=zaman();
```

نکات :

در توابع میتوان بجای استفاده از مقدار از نام تابع نیز استفاده نمود

```
Function zaman( ) : string;
Begin
zaman:= TimeToStr ( Now) ;
End ;
```

در مورد پارامترها بايد ترتيب و نوع پارامترهاي وارد شده با ترتيب و نوع پارامترهايي كه در معرفي تابع يا روال نوشته شده هماهنگ باشد

در مورد توابع از آنجا كه يك مقدار بر مي گرداند همانند يك متغير عمل كرده و مي توان آنها را برابر متغيري از نوع خودش قرار داد .

توابع بايد حتما مقدار بازگشتي داشته باشند.

زير برنامه های بدون پارامتر را میتوان بدون () نیز فراخوانی کرد مثلا تابع زمان را میتوان به صورت زیر نیز نوشت

```
Label1.caption:=zaman;
```

از اشیا و کلاسها نیز میتوان به عنوان پارامتر استفاده نمود برای استفاده از یک شیئ در زیر برنامه نوع پارامتر را از کلاس ان شیئ انتخاب میکنیم

```
Procedure testbtn ( btn : TButton);
Begin
btn . Caption := ' Test ' ;
btn . Enabled := False;
End ;
```

و برای فراخوانی ان به صورت زیر عمل میکنیم

```
Testbtn(Button1);
```

انواع پارامترها

رویه ها از ۲ نوع متغییر استفاده میکنند بنام مجازی و حقیقی

```
Equal(i) ;
```

در اینجا متغییر i متغییر حقیقی می باشد

متغییر های حقیقی متغییر هایی هستند که از برنامه در رویه ها قرار میگیرند

متغییرهای مجازی متغیرهایی هستن که هنگام اجرای رویه مقدار خود را از متغییر های حقیقی میگیرند

```
Function Equal(j : Integer) : Boolean ;
```

و در اینجا j متغییر مجازی می باشد

به متغییر هایی که به یک رویه فرستاده می شود پارامتر گویند.این پارامتر ها به ۳ دسته تقسیم می شوند

۱.ثابت const

۲.مقدار

۳.متغییر var

پارامترهای ثابت پارامترهایی هستند که در طول روند عملیات مقدار انها قابل تغییر نیست و متغییر حقیقی ان نیز ثابت است

```
procedure hello(const s:string);
begin
s:='New Delphi';
  showmessage(s);
end;
```

```
procedure TForm2.Button1Click(Sender: TObject);
var
st:string;
begin
st:='I Love Delphi';
hello(st);
ShowMessage(st);
end;
```

همانطور که میبینید مقدار پارامتر رویه از نوع ثابت بوده و با Const تعریف شده است با اجرای کد بالا با خطا موجه خواهید شد زیرا;' s:='New Delphi سعی در تغییر مقدار S دارد که با حذف این خط از برنامه برنامه اجرا خواهد شد و ۲ messagebox با متن I Love Delphi نمایش خواهد داد

پارامترهای مقدار پارامترهایی میباشند که در طول روند عملیات در رویه مقدار انها قابل تغییر می باشد ولی مقدار متغییر حقیقی ان ثابت است

```
procedure hello(s:string);
begin
s:='New Delphi';
  showmessage(s);
end;
procedure TForm2.Button1Click(Sender: TObject);
var
st:string;
begin
st:='I Love Delphi';
hello(st);
ShowMessage(st);
end;
```

با اجرای تابع فوق شما ۲ messagebox اولی با متن New Delphi و دومی با متن I Love Delphi خواهید داشت

پارامتر متغییر پارامترهایی می باشد که هم خود و هم متغییر حقیقی ان در طول روند عملیات قابل تغییر است

مثال:

```
procedure hello(var s:string);
begin
s:='New Delphi';
  showmessage(s);
end;
procedure TForm2.Button1Click(Sender: TObject);
var
st:string;
begin
st:='I Love Delphi';
hello(st);
ShowMessage(st);
end;
```

با اجرای کدهای فوق شما ۲ messagebox با عنوان New Delphi خواهید داشت

استفاده از ارایه ها به عنوان پارامتر

برای ارسال ارایه ها به توابع یا رویه ها باید انها را به صورت مقدار متغییر در اورید در این حالت یک کپی از ارایه به روال یا تابع فرستاده می شود و ۳ حالت پارامتر بر ان صدق میکند

```
Type
Intarr=array[0..100] of integer;
Procedure max(arr:intarr);
Begin
دستورات
End;
```

Forward:

قبل از فراخوانی یک تابع یا رویه باید انرا تعریف کرد در شرایطی میتوان قبل از تعریف از ان استفاده نمود برای این منظور باید از کلمه forward در انتهای تابع یا رویه استفاده نمود تا کامپایلر متوجه شود که این تابع بعدا تعریف خواهد شد

```
Functionنام تابع(پارامتر):خروجی تابع;Forward;
Procedureنام رویه(پارامتر);Froward;
```

مثال:

```
var
  Form2: TForm2;
  procedure hello(s:string);forward;
implementation

$}R *.dfm{

procedure TForm2.Button1Click(Sender: TObject);
var
st:string;
begin
st:='I Love Delphi;'
hello(st);
end;

procedure hello(s:string);
begin
  showmessage(s);
end;
```

OverLoad:

به طور معمول شما نمی توانید ۲ تابع یا رویه با نام مشابه ایجاد کنید در این حالت با خطا مواجه خواهید شد در این حالت شما باید از کلمه کلیدی OverLoad استفاده کنید با اضافه کردن کلمه OverLoad به انتهای توابع یا رویه ها میتواند هر تعداد که نیاز دارید تابع یا رویه هم نام بسازید توجه داشته باشید که پارامترهای ارسالی انتها باید متفاوت باشد

```
function Min (A,B: Integer): Integer; overload;
function Min (A,B: Int64): Int64; overload;
function Min (A,B: Single): Single; overload;
function Min (A,B: Double): Double; overload;
function Min (A,B: Extended): Extended; overload;
```

مثال:

```
function hello(s:string):string;overload;
begin
دستورات
end;

procedure hello(i:integer);overload;
begin
دستورات
end;
```

مقدار دهی اولیه

میتوان پارامترها را مقدار دهی اولیه کرد

```
procedure hello(i:string='hello');
begin
  showmessage(i);
end;

procedure TForm2.Button1Click(Sender: TObject);
begin
hello();
end;
```

با اجرای کد بالا و کلیک روی دکمه پیغام hello نمایش داده خواهد شد

توابع بازگشتی

توابع بازگشتی توابعی هستند که بطور مستقیم یا غیر مستقیم خودشان را فراخوانی میکنند توجه داشته باشید که این تعداد فراخوانی باشد محدود باشد

مثال:

```
Function fact(num:integer):integer;
Begin
```

```
If num=0 then
Fact:=1
Else
Fact:=num*fac(num-1);
End;

procedure TForm2.Button1Click(Sender: TObject);
begin
Showmessage(inttostr(fac(18)));
end;
```

قرار دادن کدها در یونیت

یکی از قابلیتهای مهم دلفی استفاده از یونیتها می باشد شما میتوانید توابع و رویه هایی که در برنامه ها زیاد با ان سرو کار دارید را درون یک unit قرار داده و به راحتی در برنامه های خود استفاده کنید برای این کار یک پروژه جدید ایجاد کنید از گزینه File/New/Unit یک Unit به پروژه اضافه کنید unit فوق را با نام Sample ذخیره کنید به قسمت InterFace یونیت رفته و با استفاده از کلمه کلیدی Uses یونیتهای مورد نیاز مانند Classesو Sysutils را به unit اضافه میکینم

```
unit sample;

interface
uses
SysUtils,Classes;

Implementation
```

النون به قسمت Implementation رفته و توابع مورد نظر را پیاده سازی میکنیم هر تابعی که پیاده سازی میکینم اعلان ان تابع را قبل از implementation قرار دهید

```
unit sample;

interface

uses
```

```
SysUtils,Classes;

 function number(str:string):string;
 function _3digit(s:string):string;

implementation

function number(str:string):string;
var
i:integer;
st:string;
begin
for I := 0 to length(str) do
  if str[i] in['0'..'9'] then
  st:=st+str[i];
  Result:=st;
end;

function _3digit(s:string):string;
var
str:string;
begin
str:=number(s);
Result:=FormatFloat('#,',StrToInt64(str));
end;
end.
```

در این یونیت ۲ تابع وجود دارد اولی بنام Number که مقداری از نوع String گرفته و اعداد انرا جدا میکند و به صورت رشته ای از اعداد به خروجی پاس میدهد و دومی تابعی بنام 3digit_ که مقداری به عنوان String دریافت و بعد با استفاده از تابع number اعداد را از متن جدا کرده و با استفاده از تابع FloatString اعداد را ۳ رقم ۳ رقم جدا کرده و به خروجی پاس میدهد یونیت Sample را ۲باره ذخیره کنید حالا نوبت به استفاده از این یونیت در برنامه می شود نام یونیت را در لیست uses پروژه قرار دهید

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls,sample;
```

حالا میتوانید از توابع داخل یونیت به طور مستقیم استفاده کنید ۲ تا Edit روی فورم قرار دهید میخواهیم با استفاده از توابع داخل یونیت کاری کنیم که Edit اولی فقط اعداد را در خود نگه دارد و Edit دومی مقدار پولی را در خود حفظ کند

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
if edit1.Text<>'' then
begin
edit1.Text:=number(edit1.Text);
edit1.SelStart:=length(edit1.Text)+1;
end;
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
if edit2.Text<>'' then
begin
  edit2.Text:=_3digit(edit2.Text);
  edit2.SelStart:=length(edit2.Text)+1;
end;
end;
```

استفاده از متد SelStart باعث می شود تا نشانگر موس همواره در انتهای اعداد قرار گیرد

*** استفاده از یونیت کلیه مزایای استفاده از زیر برنامه ها را دارند به علاوه اینکه برای استفاده از انها نیاز به کپی کردن انها از برنامه قبلی به برنامه جدید نیست زیرا فقط کافیست انها را در یک unit قرار داده و با کپی یونیت رون پوشه پروژه و افزودن نام ان به uses از کلیه توابع درون ان استفاده کنید