فایلهای دودویی:

برای استفاده از فایلهای باینری (دودویی به معنای آنکه هر داده ای در آنها ذخیره می شود، بر خلاف فایلهای متنی)

نکته دیگری مطرح می شود و آن انواع فایلهای باینری است در فایلهای باینری یا تمام داده ها از یک " نوع داده " خاص هستند مثل $TComplex, Char, Interger$ و یا اینکه ساختارهای متفاوتی در آنها ذخیره شده. در حالت اول مراحل زیر انجام می شود:

الف. تعریف متغیر از نوع فایل $var \quad < Name >: file \quad of \quad < var - type >;$

ب. نسبت دادن نام فایل به آن $AssignFile(< Name >, < Filepath >);$

پ. با یکی از دستورات $Reset$ و $Rewrite$ آن را می گشاییم دقت شود که نمی توان از $Append$ استفاده کرد.

ت. خواندن و نوشتن در و از فایل توسط دستورات $Write, Read$:

دقت شود تنها نوع داده قابل خواندن از فایل همان نوع داده $< \text{var} - type >$ است و همچنین دقت شود که از دستورات $Writeln, Readln$ نمی توان استفاده کرد:

ث. بستن فایل با دستور $CloseFile$.

مثال .

```
var
  X : file of Integer;
  I : Integer;
begin
  AssignFile(X ,'C :\Data.Int ');
  Reset (X );
  Read (X ,I );
  CloseFile(X );
end .
```

در حالت دوم مراحل زیر:

الف. تعریف متغیر نوع فایل $var \quad < Name >: file;$

ب. نسبت دادن فایل به آن $AssignFile(< Name >,< FilePath >);$

پ. باز کردن آن توسط $Reset$ **یا** $Rewrite$

$Reset(< Name >,< RecordSize >);$

$Rewrite(< Name >,< RecordSize >);$

در این مورد $(< recordSize >)$ **عددی است که طول حداقل واحد حافظه قابل خواندن از فایل را به بایت مشخص می کند معمولاً در استفاده آن را 1 قرار می دهیم.**

ت. خواندن و نوشتن با دستورات $BlockWrite, BlockRead$

$BlockRead(< Name >,< var-name >,< size >);$

$Blockwrite(< Name >,< var-name >,< size >);$

این دستورات به اندازه $< RecordSize > * < Size >$ **بایت از فایل خوانده و آنرا به محل متغیر** $< var-name >$ **در حافظه کپی می کنند و یا بالعکس از مکان حافظه مقدار ذکر شده! بایت خوانده و در فایل می نویسند.**

ث. در نهایت بستن فایل ;(< *Name* >)*CloseFile*

مانند

```
var
  X  : file;
  I : Integer;
  J : Extended;
begin
  Assign(X ,'C :\Data.Dat');
  Reset (X ,1);
  BlockRead (X ,T ,Sizeof (I ));
  BlockRead (X ,J ,Sizeof (J ));
  CloseFile (X );
end .
```

البته برای کار با فایلهای باینری روش بسیار بهتری وجود دارد و آن کار با شیئی *TFileStrea m* **است.**

در روش کار با شیئی *TFileStream* **مراحل بعد را داریم:**

الف. تعریف متغیر نوع *FileStream* : *var < Name >:TFileStream;*

ب. گشودن و نسبت دادن فایل ;**(**< *FilwPath* >,< *mode* >**)**;*< Name >:TFileStream .Create*

بجای < *mode* > **از یکی از عبارت های** *fmOpenReadWrite* , *fmOpenWrite* , *fmOpenRead* **و** *fmCreate* **استفاده می کنیم. اینها همه اعضای یک** *Enum* **هستند اولی باز کردن فقط برای خواندن دومی باز کردن برای نوشتن و سومی باز کردن برای هم خواندن و هم نوشتن و آخری برای ایجاد فـایلی کـه وجـود ندارد استفاده می شود.**

پ. خواندن و نوشتن با دستورات

$<Name>.Read\ <var-name>,<count>);$

$<Name>Write\ <var-name>,<count>);$

که کاملاً مشابه $BlockWrite, BlockRead$ هستند با این تفاوت که دیگر $<RecordSize>$ همواره **1** است. و تعداد $Count$ بایت بین فایل و حافظه جابجا می گردد.

ث. در این نوع کار با فایل نیازی به هیچ کاری برای بستن نیست و فایل با اتمام برنامه خودبخود بسته می شود.

در کار با فایلهای باینری پیش می آید که درون فایل جابجا شویم به این معنا که اطلاعات را بصورت تصادفی از جایگاه های متفاوت فایل بخوانیم $(RandomAccess)$ در اینگونه موارد از توابع $Seek$ استفاده می شود که در هنگام عدم استفاده از $TFileStream$ دستور به شکل زیر :

$Seek(<Name>,<Position>);$ که $Position$ بر حسب $<RecordSize>$ از ابتدای فایل

داده می شود (یا بر حسب $(<var-type>)\ Size\ Of$ استفاده شده، یعنی اندازه نوع داده استفاده شده) و در هنگام استفاده از $TFileStream$:

$<Name>: Seek(<Position>,<Origin>);$

که $<Position>$ بر حسب بایت از ابتدا، محل جاری و یا انتهای فایل محاسبه می شود وقتی $<Origin>$ به ترتیب برابر $SoFromEnd, SoFromCurrent, SoFromBogining$ باشد.