فصل اول

فایل‌ها

هدف‌های رفتاری: فراگیر پس از پایان این فصل، خواهد توانست :

۱ــ انواع فایل‌های ترتیبی و تصادفی را شرح دهد و با انواع فایل‌های ترتیبی و تصادفی کار کند.

۲ــ چگونگی استفاده از شماره‌ی فایل را شرح دهد و به کار ببرد.

۳ــ از دستورهای #Print، #Write، #Read، # Get و #Put در فایل‌ها استفاده کند.

۴ــ کنترل‌های مربوط به فایل را در برنامه‌های خود به‌کار گیرد.

۵ــ اهمیت کاربرد فایل‌ها را بیان کند.

۶ــ چگونگی باز کردن فایل‌ها را بیان کرده و انجام دهد.

۷ــ فایل‌ها را در سه حالت مختلف باز کند.

۱ــ۱ــ آشنایی با مفهوم پایداری

برای اینکه برنامه‌ای بتواند اطلاعات را از جلسه‌ای به جلسه‌ی کاری دیگری نگه دارد، باید توانایی ذخیره‌سازی داده‌ها روی دیسک سخت را داشته باشد. در غیر این صورت، هنگامی که برنامه‌ی کاربردی قطع می‌شود، تمام داده‌های برنامه که در حافظه اصلی هستند از بین می‌روند.

بنابراین، برای اینکه داده‌ها پایدار باشند، باید برنامه توانایی ذخیره و بازیابی داده‌ها در دیسک سخت را داشته باشد.

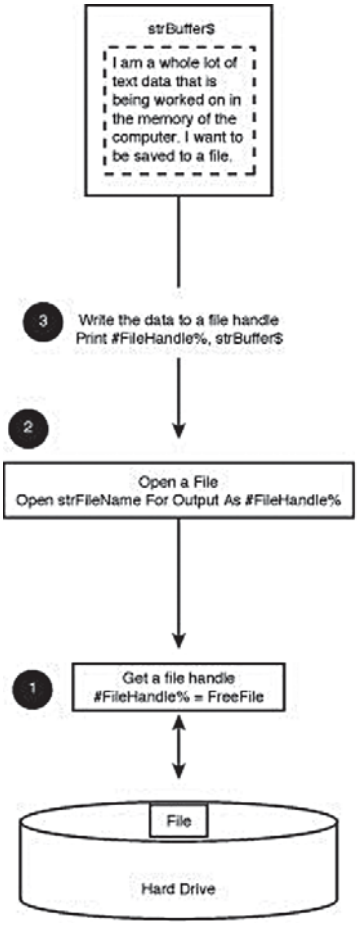
می‌توان با چندین روش، عملیات ذخیره و بازیابی داده‌ها را انجام داد. از فایل دودویی یا متنی می‌توان برای ذخیره‌ی اطلاعات با اندازه و قالب متفاوت استفاده کرد. عملیات خواندن و نوشتن می‌تواند شامل تغییرات جزیی در رجیستری ویندوز یا اطلاعات پیچیده‌تری در بانک اطلاعاتی باشد.

۲ــ۱ــ کار کردن با فایل‌ها برای ذخیره و بازیابی داده‌ها

داده‌ها در حافظه‌ی اصلی و فایل‌ها روی حافظه جانبی (دیسک‌ها) ذخیره می‌شوند. هرگز برنامه‌ها به‌طور مستقیم با یک فایل روی دیسک کار نمی‌کنند.

برنامه از سیستم عامل می‌خواهد که واسطی بین دیسک و برنامه ایجاد کند (به عبارت دیگر، داده‌های مورد نیاز برنامه به کمک پردازنده و سیستم عامل به حافظه‌ی اصلی بارگذاری می‌شوند و سپس برنامه اجرا می‌شود).

موقعیت (محل) یک فایل روی دیسک را می‌توان با به دست آوردن file handle از طریق سیستم عامل، پیدا کرد. از تابع FreeFile برای به‌دست آوردن شماره‌ی فایل از طریق سیستم عامل استفاده کنید. بعد از به‌دست آوردن این شماره، دستور Open را برای دسترسی به فایل برای عملیات خواندن و نوشتن به کار ببرید. برای نوشتن در یک فایل، از دستور Print (یا Write) و برای خواندن خطوطی از داده‌ها از فایل روی دیسک، از دستور Line Input استفاده کنید. شکل ۱ــ۱، این مفهوم را شرح می‌دهد.



شکل ۱ــ۱ــ نوشتن در یک فایل، برعکس خواندن از فایل است. برای این کار نیاز به دسترسی به File handle و استفاده از دستور Open دارید.

فایل‌ها سه نوع هستند : ترتیبی (sequential)، تصادفی (random) و دودویی (binary). فایل ترتیبی ساده‌ترین نوع فایل است اما دارای معایبی است که یکی از آن‌ها کند بودن کار با این نوع فایل‌ها است. کار با فایل‌های تصادفی سرعت بیشتری دارد اما پیچیدگی بیشتری را به برنامه تحمیل می‌کند. فایل‌های دودویی نوع خاصی از فایل‌های تصادفی هستند که در ادامه‌ی فصل با آن‌ها آشنا خواهید شد.

۳ــ۱ــ دستور Open

از دستور Open می‌توان برای ذخیره و بازیابی داده‌ها از یک فایل روی دیسک استفاده کرد. شکل کلی دستور Open به صورت زیر است :

Open *FilePath* [For *Mode*] [Access *AcessType* [*LockType*] As [#]*FileNumber* [Len=*CharInBuffer%*]

در این دستور :

- Open نام دستور است.
- *FilePath* محل دقیق فایل برای خواندن یا ذخیره کردن که شامل درایو و مسیر است.
- For کلید واژه‌ای است که حالت فایل به دنبال آن ارایه می‌شود.
- *Mode* حالت دسترسی به فایل است (جدول ۱ــ۱).

جدول ۱ــ۱ــ حالت‌های دسترسی به فایل

حالت	شرح
Appentd	اضافه کردن داده‌ها به انتهای یک فایل ترتیبی موجود. در صورتی که فایل موجود نباشد، آن را ایجاد خواهد کرد.
Binary	فایل را به صورت دودویی باز می‌کند (بیت‌ها و بایت‌ها). در صورتی که فایل وجود نداشته باشد، آن را ایجاد خواهد کرد.
Input	فایل ترتیبی را برای خواندن باز می‌کند.
Output	فایل ترتیبی را برای نوشتن باز می‌کند. در صورتی که فایل وجود نداشته باشد، آن را ایجاد خواهد کرد.
Random	فایل را برای دسترسی تصادفی باز می‌کند. برای ذخیره‌سازی رکوردی مورد استفاده قرار می‌گیرد. در صورتی که فایل وجود نداشته باشد، آن را ایجاد خواهد کرد.

● Access کلید واژه‌ای اختیاری است که نوع دسترسی بعد از آن آورده می‌شود.

● *AccessType* انتخابی بین Read، Write یا Read Write است.

● *LockType* (اختیاری) تعیین می‌کند که آیا همزمان با این‌که برنامه روی فایل کار می‌کند، دیگران می‌توانند فایل را بخوانند. مقادیری که پشتیبانی می‌کند، عبارتند از : Shared، Lock Read، Lock Write و Lock Read Write.

به‌طور عادی، این مقادیر مربوط به استفاده از فایل داده‌ها در محیط شبکه هستند. هنگامی که مقدار Shared انتخاب شود، کاربران می‌توانند به فایل دسترسی داشته باشند. اگر مقدار تعیین شده، Lock Read باشد، کاربران فقط می‌توانند داده‌ها را از فایل بخوانند و در مورد سایر مقادیر نیز به همین ترتیب است. این مقادیر را می‌توان در محیط PC منفرد نیز به‌کار برد ولی بیشتر در محیط شبکه‌ای متداول است.

● As کلید واژه‌ای که تعیین می‌کند شماره‌ی فایل بعد از آن ارایه می‌شود.

● *FileNumber* شماره‌ی file handle است.

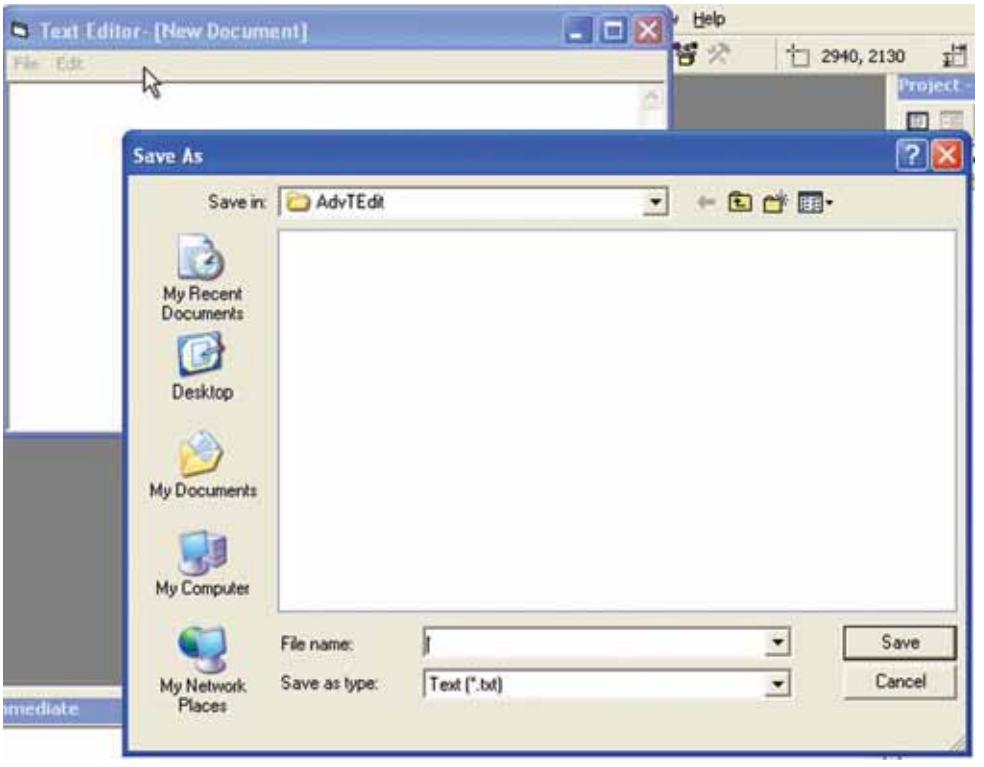
● Len کلید واژه‌ی اختیاری است که قبل از پارامتر طول رکورد قرار می‌گیرد.

● *CharInBuffer%* یک گزینه‌ی اختیاری است که طول رکورد فایلی که برای دسترسی تصادفی باز شده است را تعیین می‌کند.

یک فایل را برای نوشتن به صورت زیر باز کنید :

```
FileHandle% = FreeFile
Open strFileName For Output As # FileHandle%
```

مثال ۱ــ۱ــ ایجاد یک ویراستار متن: پروژه‌ای ایجاد کنید که دارای دو منوی File و Edit باشد. منوی File دارای گزینه‌های Save برای ذخیره و ایجاد فایل متنی و Open برای بازکردن فایل خواهد بود. برای استفاده از برنامه، هنگام ذخیره‌ی داده‌ها، روی گزینه‌ی Save از منوی File کلیک کنید. در کادر محاوره‌ای که ظاهر می‌شود، نام فایل را وارد کرده و مسیر آن را وارد کنید. سپس روی دکمه‌ی Save در کادر محاوره‌ای کلیک کنید تا داده‌ها ذخیره شوند (شکل ۲ــ۱).



شکل ۲ــ۱ــ استفاده از کادر محاوره‌ای، روش ساده‌ای برای ایجاد نام فایل و محلی برای داده‌هاست.

کد زیر، روال رویداد Click() گزینه‌ی save را نشان می‌دهد. این روال یک فایل را با دستور Open (خط ۲۸) باز می‌کند و محتوای کادر متن را با استفاده از متد Print ذخیره می‌کند (خط ۳۴). روال رویداد، فایل را با استفاده از دستور Close می‌بندد (خط ۴۰). دستور Close شماره‌ی file handle را به عنوان یک آرگومان می‌گیرد.

```
01 Private  Sub itmSave_Click()
02 Dim strFileName As String `String of file to open
03 Dim strText As String `Contents of file
04 Dim strFilter As String `Common Dialog filter string
05 Dim strBuffer As String `String buffer variable
06 Dim FileHandle% `Variable to hold file handle
07
```

```
08 `Set the Common Dialog filter
09 strFilter = "Text (*.txt)|*.txt| All Files (* .*)|*.*"
10 cdMain.Filter = strFilter
11
12 `Open the common dialog in save mode
13 cdMain.ShowSave
14
15 `Make sure the retrieved filename is not a blank string
16 If cdMain.filename <>"" Then
17 `If it is not blank, open the file
18 strFileName = cdMain.filename
19
20 `Assign a value to the text variable
21 strText = txtMain.Text
22
23 `Get a free file handle and assign it to the file
24 `handle variable
25 FileHandle% = FreeFile
26
27 `Open a file for writing
28 Open  strFileName For Output As # FileHandle%
29
30 `Set an hour glass pointer just in case it takes a while
31 MousePointer = vbHourglass
32
33 `Do the write
```

```
34 Print # FileHandle%, strText
35
36 `Reset the pointer to the Windows default.
37 MousePointer = vbDefault
38
39 `Close the file when completed
40 Close # FileHandle%
41 End If
42
43 End Sub
```

نکته: به خاطر داشته باشید که بعد از پایان کار با فایل، آن را ببندید (دستور Close). این دستور شماره‌ی File handle را از حافظه خارج می‌کند.

۱ــ۳ــ۱ــ طول رکورد

آرگومان Len = CharInBuffer هنگام کار با فایل‌های تصادفی اهمیت می‌یابد. هنگامی که ویژوال بیسیک از فایلی که برای دسترسی تصادفی باز شده است، اطلاعات را می‌خواند (یا در آن می‌نویسد)، باید طول رکورد (Record) را بداند. هر فایل تصادفی در واقع مجموعه‌ای از N قسمت است که هر قسمت معادل CharInBuffer طول دارد. کار با فایل‌های تصادفی بسیار شبیه کار با آرایه‌هاست و اگر از دستور Option Base 1 استفاده کنید بین آن‌ها یک تناظر یک به یک ایجاد خواهد شد، چون شماره‌ی رکوردها هم از ۱ شروع خواهد شد.

۲ــ۳ــ۱ــ تخصیص شماره‌ی فایل

در Visual Basic این امکان وجود دارد که به‌طور همزمان چندین فایل باز داشته باشید، مشروط بر اینکه هر فایل شماره‌ی خاص خود را داشته باشد. به این منظور همیشه باید شماره‌ی فایل‌های باز را بدانید و این کار بسیار مشکلی است. اما Visual Basic برای این مشکل راه حلی دارد : تابع FreeFile(). این تابع، اولین شماره‌ی فایل آزاد را برمی‌گرداند. با استفاده از این تابع

همواره اطمینان خواهید داشت که شماره‌ی فایل‌ها تکراری نخواهند بود. شکل استفاده از تابع FreeFile() چنین است :

```
FreeFile [ (intRangeNumber) ]
```

پارامتر اختیاری intRangeNumber اجازه می‌دهد تا محدوده‌ی شماره‌ی فایل را (1-255 یا 256-511) مشخص کنید. محدوده‌ی پیش‌فرض 1-255 است. اغلب برنامه‌نویسان Visual Basic از همین محدوده استفاده می‌کنند چون به ندرت پیش می‌آید که یک برنامه در آن واحد بیش از ۲۵۶ فایل باز داشته باشد. در این حالت حتی نوشتن پرانتزها هم ضروری نیست. در دستورهای زیر، ابتدا یک شماره‌ی فایل مشخص شده و سپس فایل موردنظر با این شماره باز می‌شود :

```
intFileNumber = FreeFile()
Open "AccPay.Dat" For Output As intFileNumber
```

با استفاده از FreeFile() می‌توانید مطمئن شوید که حتی در برنامه‌های کوچک، دو فایل با شماره‌ی مشابه وجود نخواهند داشت. دو دستور فوق را می‌توان در یک دستور نیز خلاصه کرد :

```
Open "AccPay.Dat" For Output As FreeFile()
```

با این روش، امکان اینکه شماره‌ی فایل باز شده را بدانید، وجود ندارد.

۴ــ۱ــ دستور Close

هر فایلی که باز شده باید بعد از استفاده بسته شود. دستور بستن فایل ساده است :

```
Close# intFileNumber1[,intFileNumber2] [,... intFileNumberX]
```

تعداد فایل‌هایی که می‌توانید در این دستور قید کنید محدودیتی ندارد و اگر هیچ شماره‌ای قید نشود تمام فایل‌های باز، بسته خواهند شد. در کد زیر، ابتدا دو فایل (یکی برای خواندن و دیگری برای نوشتن) باز شده و سپس بسته می‌شوند.

```
Dim intReadFile As Integer, intWriteFile As Integer
intReadFile = FreeFile      'Get first file #
Open "AccPay.Dat" For Input As intReadFile
IntWriteFile = Freefile     'Get next file #
Open "AccPayOut.Dat" For Output As intWriteFile
Close intReadFile
```

```
Close intWriteFile
```

اگر فایلی بسته نشود، احتمال صدمه به آن وجود دارد. بنابراین به محض اینکه کارتان با یک فایل تمام شد، در اولین فرصت آن را ببندید. بهترین روش (علاوه بر بستن تک‌تک فایل‌ها) آن است که در پایان برنامه با دستور ساده‌ی زیر تمام فایل‌های باز را به یکباره ببندید :

```
Close
```

۵ــ۱ــ کار با فایل‌های ترتیبی

حال که با کلیات فایل (مانند باز کردن، بستن، نوع دسترسی و قفل کردن فایل‌ها) آشنا شدید، بهتر است با چند مثال خواندن ــ نوشتن فایل‌های ترتیبی را بیشتر تجربه کنیم.

فایل ترتیبی یعنی فایلی که فقط به صورت متوالی می‌توان با آن کار کرد ؛ فایل‌های ترتیبی را باید از اول تا آخر فایل خواند یا نوشت. خواندن / نوشتن بزرگ‌ترین نقطه ضعف فایل‌های ترتیبی است. برای استفاده از این قبیل فایل‌ها باید تمام فایل را خواند. مثلاً اگر فایلی ۱۰۰۰ بایت باشد و فقط بخواهید ۱ بایت آن را تغییر دهید، باید تمام بایت‌های دیگر را هم بخوانید و دوباره بنویسید. فایل‌های ترتیبی برای ذخیره کردن فایل‌های متنی کوچک که در آن‌ها سرعت چندان مهم نیست، مناسب هستند.

۱ــ ۵ ــ۱ــ دستور #Print

قبل از استفاده از هر فایلی باید آن را باز کنید. بعد از باز کردن فایل، می‌توانید اطلاعات را در آن بنویسید. یکی از روش‌های نوشتن در فایل استفاده از دستور #Print است. با این دستور فقط در فایل‌های ترتیبی می‌توان نوشت :

```
Print# intFileNumber, [OutputList]
```

در این دستور، intFileNumber شماره‌ی فایل موردنظر و OutputList یکی از اقلام زیر است :

```
[Spc(intN1)| Tab (intN2)]] [Expression] [charPos]
```

طرز کار توابع Spc() و Tab() در دستور #Print مشابه دستور Print (در برنامه‌سازی ۱ با این دستور و توابع آشنا شده‌اید) است. توجه کنید که از هر دو تابع همزمان نمی‌توانید استفاده کنید. در جدول ۲ــ۱ توضیحات بیشتری درباره‌ی عناصر OutputList ملاحظه می‌کنید.

جدول ۲ــ۱ــ عناصر لیست خروج دستور Print #

عنصر	مفهوم
Spc(intN1)	به مقدار intN1 بین خروجی‌ها فاصله می‌اندازد.
Tab (intN2)	مکان ظاهر شدن خروجی را تعیین می‌کند. intN2 شماره‌ی ستون را مشخص می‌کند. برای نوشتن در مناطق چاپی (به فواصل ۱۴ کاراکتر) از دستور Tab بدون آرگومان استفاده کنید.
Expression	عبارت عددی یا رشته‌ای
Charpos	محل ظاهر شدن دستور Print بعدی. با ; دستور Print بعدی از ادامه‌ی همین خط، نوشتن خروجی را ادامه خواهد داد. اگر این عنصر حذف شود، دستور Print بعدی از ابتدای خط بعد شروع خواهد شد.

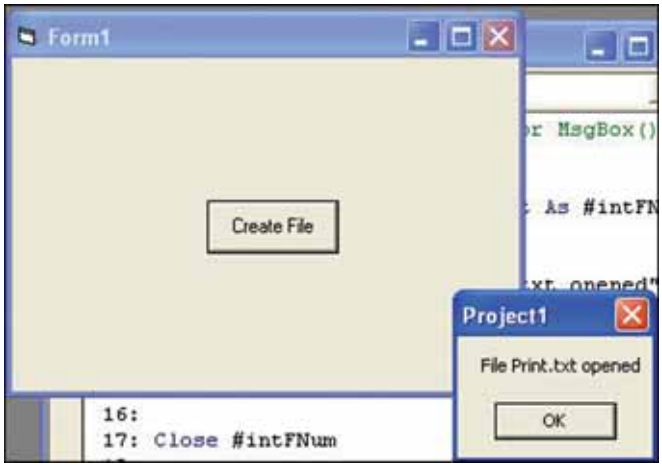
مثال ۲ــ۱ــ در کد زیر، روالی را مشاهده می‌کنید که فایل Print.txt را باز کرده، اعداد ۱ تا ۶ را در آن فایل می‌نویسد و سپس فایل را می‌بندد.

```
Private Sub cmdFile_Click()
    Dim intCtr As Integer   'Loop counter
    Dim intFNum As Integer    'File number
    intFNum = FreeFile
    Open "C:\Print.txt" For Output As #intFNum
    MsgBox "File Print.txt opened"
    For intCtr = 1 To 6
        Print #intFNum, intCtr      'Write the loop counter
        MsgBox "Writing a "& cstr(intCtr) & "to Print.txt"
    Next intCtr
    Close #intFNum
    MsgBox "File Print.txt closed"
End Sub
```

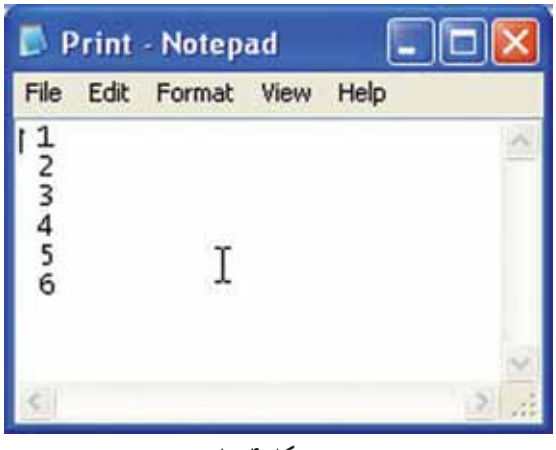
برنامه را اجرا کنید. این برنامه در هر مرحله یک کادر پیام نمایش خواهد داد : هنگام باز شدن

فایل، هنگام نوشتن اعداد در فایل و هنگام بسته شدن فایل. برای اینکه مطمئن شوید این روال، کار خود را به درستی انجام داده است، در برنامه‌ی NotePad ویندوز فایل Print.txt را باز کنید. اعداد ۱ تا ۶ را باید در این فایل مشاهده کنید :

1
2
3
4
5
6



شکل ۳ــ۱



شکل ۴ــ۱

اما اگر نتوانید فایل‌ها را بخوانید، نوشتن آن‌ها چه فایده‌ای دارد؟ در مطالب بعد نشان می‌دهیم که چگونه می‌توانید محتویات یک فایل را بازیابی کنید.

۲ــ ۵ ــ۱ــ دستور #Input

بعد از نوشتن اطلاعات در فایل، باید بتوانید آن‌ها را دوباره بازیابی کنید. دستور خواندن فایل‌های ترتیبی، دستور #Input است. خواندن یک فایل ترتیبی باید دقیقاً به همان ترتیب نوشتن آن صورت گیرد :

```
Input #intFileNumber, Variable1 [, Variable2] [,... VariableN]
```

دستور #Input هم به یک شماره‌ی فایل نیاز دارد. این دستور باید دقیقاً متناظر با دستور #Print همان فایل باشد. دستور زیر پنج مقدار را از فایل باز شده می‌خواند و آن‌ها را در متغیرهای V1 تا V5 قرار می‌دهد :

```
Input #intFileNumber V1, V2, V3, V4, V5
```

نوع داده‌ی این پنج متغیر باید با دستور #Print که این مقادیر را در فایل نوشته است، مطابقت داشته باشد. در غیر این صورت، دستور #Input قادر به خواندن داده‌ها نخواهد بود.

مشکلاتی که در انطباق دستورات #Print و #Input وجود دارد، ما را به یک دستور کلی‌تر و قابل انعطاف‌تر می‌رساند : دستور Write.

۳ــ ۵ ــ۱ــ دستور #Write

دستور #Write فرمان دیگری است برای نوشتن در فایل‌های ترتیبی. دستورات #Write و #Print تفاوت کمی با هم دارند : داده‌هایی که با #Write در فایل نوشته می شوند با کاما (،) از هم جدا خواهند شد. این دستور هنگام نوشتن رشته آن را درون زوج گیومه (") قرار خواهد داد و برای نوشتن تاریخ از # استفاده می‌کند. مقادیر Boolean را به صورت #TRUE# و #FALSE# می‌نویسد. داده‌های null و خطا را به همین صورت #NULL# و #Error *errorcode*# خواهد نوشت. *errorcode* کد تولید شده به وسیله‌ی خطای رخ داده است.

از آن‌جایی که داده‌ها با #Write به وسیله‌ی کاما از هم جدا خواهند شد، انطباق #Input و #Write ضروری نیست. بنابراین برای سهولت کار، سعی کنید از #Write به جای #Print استفاده کنید. شکل کلی دستور #Write چنین است :

```
Write#intFileNumber , [OutputList]
```

که در آن OutputList فهرست متغیرهایی است که باید در فایل intFileNumber نوشته شوند.

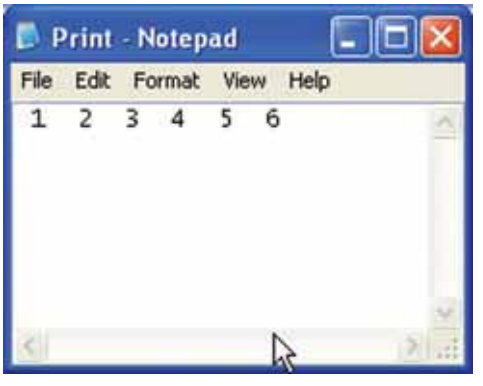
مثال ۳ــ۱ــ به کد زیر توجه کنید. در این برنامه بعد از دستور Print# از ; استفاده شده است، بنابراین اعداد پشت سر هم در فایل نوشته خواهند شد.

```
Private Sub cmdFile_Click()
    Dim intCtr As Integer   'Loop counter
    Dim intFNum As Integer   'File number
    intFNum = FreeFile
    Open "C:\Print.txt" For Output As #intFNum
    MsgBox "File Print.txt opened"
    For intCtr = 1 To 6
        Print # intFNum, intCtr;      Notice semicolon!
        MsgBox "Writing a "& cstr(intCtr) & "to Print.txt"
    Next intCtr
    Close # intFNum
    MsgBox "File Print.txt closed"
End Sub
```

خروجی این فایل چنین خواهد بود :

1 2 3 4 5 6

فاصله‌ای که بین اعداد ملاحظه می‌کنید علامت مثبت اعداد مزبوراست که به قرینه حذف شده است. دستور Print# علامت اعداد را هم در فایل چاپ خواهد کرد.



شکل ۵ ــ ۱

نوشتن و خواندن فایل‌ها به یکدیگر وابسته‌اند و هر دو عمل می‌توانند در یک برنامه انجام شوند. در کد زیر، برنامه‌ای را مشاهده می‌کنید که دو روال جداگانه عملیات نوشتن و خواندن فایل را نشان داده است.

```
1: Private Sub cmdFileOut_Click()
2:     ‘Create the sequential file
3:     Dim intCtr As Integr          ‘Loop counter
4:     Dim intFNum As Integer    ‘File number
5:     intFNum = FreeFile
6:
7:     Open “Print.txt” For Output As #intFNum
8:
9:     For intCtr = 1 To 6
10:    Print # intFNum, intCtr;      ‘Write the loop counter
11: Next intCtr
12:
13: Close # intFNum
14: End Sub
15:
16: Private Sub cmdFileIn_Click()
17: ‘Read the sequential file
18: Dim intCtr As Integer         ‘Loop counter
19: Dim intVal As Integer         ‘Read value
20: Dim intFNum As Integer    ‘File number
21: Dim intMsg As Integer       ‘MsgBox()
22: intFNum = FreeFile
23: Open “Print.txt” For Input As # intFNum
24:
```

```
25: For intCtr = 1 To 6
26:     Input # intFNum, intVal
27:     'Display the results in the lmmediate windows
28:     intMsg = MsgBox("Retrieved a "& intVal &"from Print.txt")
29: Next intCtr
30:
31: Close # intFNum
32: intMsg = MsBox("The Print.txt file is now closed.")
33: End Sub
```

اکنون کد زیر را درنظر بگیرید. در این برنامه برای نوشتن در فایل از دستور #Write استفاده شده است.

```
1: Private cmdFile_Click()
2:      Dim intCtr As Integer         'Loop counter
3:      Dim intFNum As Integer     'File number
4:      intFNum = FreeFile
5:
6:      Open "c:\Write.txt" For Output As #intFNum
7:
8:      For intCtr = 1 To 5
9:              Write # intFNum, intCtr;      'Write the loop counter
10: Next intCtr
11:
12: Close # intFNum
13: End Sub
```

برنامه را اجرا کرده و سپس فایل ایجاد شده (Write.txt) را باز کنید. به تفاوت خروجی دستورات #Write و #Print دقت کنید :

```
1,2,3,4,5,
```

اگر در پایان دستور #Write از; استفاده نکنیم، هر عدد در یک خط نوشته خواهد شد و دیگر از کاما هم خبری نخواهد بود. چون در این حالت دیگر وجود کاما برای جدا کردن داده‌ها ضرورتی ندارد. (در این حالت دستورات #Write و #Print شبیه یکدیگر خواهند شد.)

۴ــ ۵ ــ۱ــ بازیابی داده‌ها با دستورهای Input

بازیابی داده‌ها از دیسک خیلی شبیه به نوشتن داده‌ها در آن است. تنها تفاوت این است که به جای استفاده از حالت‌های Append، Output، Binary یا Random، حالت Input را به کار ببرید. همچنین به جای استفاده از متد Print یا Write، با استفاده از دستور Line Input داده‌های فایل را خط به خط بخوانید. شکل کلی دستور Line Input به صورت زیر است :

```
Line Input #FileHandle, strBuffer
```

در این دستور :

- Line Input کلید واژه‌های دستور هستند.
- FileHandle شماره‌ی فایل باز است.
- StrBuffer رشته‌ای است که دستور، داده‌های بازیابی شده را در آن قرار می‌دهد.

فایل‌های متنی ساده به صورت خط به خط در دیسک ذخیره می‌شوند. اگر متنی را در NotePad وارد کرده و کلید Enter را فشار ندهید، یک خط وارد کرده‌اید. هر بار که کلید Enter را فشار دهید، ویژوال بیسیک رشته‌ی chr(13)&chr(10) را به کادر متن اضافه می‌کند تا پایان خط را مشخص کند. هنگامی که این خط را ذخیره می‌کنید، این نویسه‌ها نیز در فایل نوشته می‌شوند. ویژوال بیسیک دارای ثابت vbCrLF برای این رشته است. دستور Line Input نویسه‌های داخل فایل را تا زمانی که به vbCrLF برسد می‌خواند. در پایان خط، دستور نویسه‌ها را به آرگومان بافر ارسال می‌کند و از vbCrLF صرف نظر می‌کند.

برای پیمایش تمام خطوط فایل، از حلقه‌ی Do While استفاده کنید. از تابع EOF() ویژوال بیسیک برای تعیین اینکه آیا به انتهای فایل رسیده‌ایم، استفاده کنید. این تابع، شماره‌ی فایل را به عنوان آرگومان دریافت می‌کند. تا زمانی که به انتهای فایل نرسیده‌اید، دستور Line Input به خواندن خطوط فایل در داخل حلقه‌ی Do While ادامه می‌دهد.

مثال ۴ــ۱ــ کد زیر، روال رویداد مربوط به گزینه‌ی Open از منوی File پروژه‌ی مثال ۱ــ۱ را نشان می‌دهد. کاربر برای بازکردن فایل در داخل ویراستار متن، روی این گزینه کلیک می‌کند. روال رویداد از یک کادر محاوره‌ای برای فراهم کردن امکان شناسایی فایلی که باید باز شود،

استفاده می‌کند.

```
01 Private  Sub itmOpen_Click()
02 Dim strFileName As String `String of file to open
03 Dim strText As String `Contents of file
04 Dim strFilter As String `Common Dialog filter string
05 Dim strBuffer As String `String buffer variable
06 Dim FileHandle% `Variable to hold file handle
07
08 `Set the Common Dialog filter
09 strFilter = "Text (*.txt)|*.txt| All Files (*.*)|*.*"
10 cdMain.Filter = strFilter
11
12 `Open the common dialog
13 cdMain.ShowOpen
14
15 `Make sure the retrieved filename is not a blank string
16 If cdMain.filename <>"" Then
17
18 `If it is not blank, open the file
19 strFileName = cdMain.filename
20
21 `Get a free file handle and assign it to the file
22 `handle variable
23 FileHandle% = FreeFile
24
25 `Open the file
26 Open strFileName For Input As #FileHandle%
```

```
27
28 `Make the mouse pointer an hourglass
29 MousePointer = vbHourglass
30
31 `Traverse the lines of the file
32 Do While Not EOF(FileHandle%) `Check for end of file
33
34 `Read a line of the file
35 Line Input #FileHandle%, strBuffer.
36
37 `Add the line from the Output buffer
38 strText = strText & strBuffer & vbCrLf
39 Loop
40
41 `Change the mousepointer back to the arrow
42 MousePointer = vbDefault
43
44 `Close the file when completed
45 Close #FileHandle%
46
47 `Assign the retrieved text to the text box
48 txtMain.Text = strText
49
50 `Put the filename in the form caption
51 frmMain.Caption = "Text Editor- ["& strFileName &"]"
52 End If
53 End Sub
```

۶ــ۱ ــ کار با فایل‌های تصادفی

با آن که درباره‌ی فایل‌های ترتیبی بسیار صحبت کردیم، ولی باید بگوییم که برنامه‌نویسان Visual Basic به ندرت از این نوع فایل‌ها استفاده می‌کنند و بیشتر تمایل دارند با فایل‌های تصادفی (random access) کار کنند. فایل تصادفی، فایلی است که در هر نقطه از آن (بدون رعایت ترتیب) می‌توان نوشت یا از آن خواند. امروزه با وجود کنترل‌های داده (Data Controls) دیگر حتی این نوع فایل‌ها نیزکاربرد وسیع خود را (برخلاف گذشته) از دست داده‌اند. دلیل اولیه‌ی اهمیت فایل‌های تصادفی، معرفی نوع جدیدی از انواع داده است : نوع داده‌ی تعریف شده به وسیله‌ی کاربر (user-defined data type). این نوع داده، همانطور که از نام آن پیداست، برخلاف انواع داده‌ی ذاتی Visual Basic به وسیله‌ی کاربر تعریف می‌شود.

در مطالبی که تاکنون بیان کرده‌ایم از انواع داده‌های ذاتی ویژوال بیسیک مثل Intger، String و Double استفاده کرده‌ایم. یکی از ویژگی‌های ویژوال بیسیک، فراهم نمودن امکان ایجاد نوع داده‌ی سفارشی است که به نام نوع داده تعریف شده به وسیله‌ی کاربر (user-defined Data Type یا به اختصار UDT) معروف است. می‌توان UDT را به عنوان متغیری فرض کرد که از چندین بخش ایجاد شده است و هر بخش را می‌توان چندین بار در برنامه به کار برد.

برنامه‌ای را در نظر بگیرید که نام قطعات موسیقی و آهنگساز هر قطعه را ذخیره می‌کند. اگر بخواهید برای هر قطعه و آهنگساز، متغیر خاصی را تعریف کنید، بخش اعلان متغیرها به صورت زیر خواهد بود :

```
01 Public g_ ComposerOne As String
02 Public g_PieceOne As String
03
04 Public g_Composer Two As String
05 Public g_ PieceTwo As String
06
07 Public g_ComposerThree As String
08 Public g_PieceThree As String
```

این قبیل کدنویسی ممکن است برای برنامه کوتاه مناسب باشد ولی اگر تعداد قطعات موسیقی که می‌خواهیم ذخیره کنیم، بیشتر شود، این روش، مناسب نخواهد بود. یک راه‌حل ساده برای این