파일 다루기

텍스트 파일이란

요즘에는 파일을 다루는 작업을 하지 않는 프로그램이 거의 없다.
파일 File 이란 정보를 기록하는 디스크나 기타 정보 저장 미디어 안에 있는 한 부분이며 이름이 붙어있다.

모든 파일은 자신들만의 고유한 형식이 있으며 이런 파일 고유 형식의 세부 사항은 파일에 데이터를 인코딩(코드로 기록) 하는 방식에 따라 결정된다. 가장 많이 사용되는 파일 형식은 텍스트 파일 Text File 형식이다.

텍스트 파일 Text File 에는 줄을 하나 이상 넣을 수 있다. 그리고, 각 줄에는 문자들을 얼마든지 넣을 수 있다. 텍스트 파일에서 한 줄이 끝나면 특별한 문자인 줄-끝남 end-of-line 문자로 표시한다. 그리고 파일의 마지막은 파일 끝남 end-of-file 문자로 끝낸다. 텍스트 파일은 메모장 노트패드나 여러 텍스트 파일 편집기에서 열고 편집할 수 있다.

텍스트 파일은 데이터들이 결합되어 묶여서 하나의 공동 이름이 붙은 파일이다. 텍스트 파일 이름은 대체로 .txt 라는 확장자를 가진다. 하지만 확장자가 .txt 라고해서 반드시 텍스트 파일이라는 보장은 없다. 물론 텍스트 파일에 .txt 가 아닌 다른 확장자를 붙일 수도 있다. 표준으로 사용되는 텍스트 파일 확장자로는 .ini, log, .inf, .dat, .bat 등이 있다. 우리는 txt 파일 확장자를 사용하기로 한다.

텍스트 파일의 장점

- 작은 크기와 다양성을 포함합니다. 킬로바이트 또는 메가 바이트는 다른 형식으로 저장된 동일한 데이터보다 작으며 이메일 또는 디스크를 통해 신속하고 대량으로 교환 할 수 있습니다.
- 대부분 기본 소프트웨어를 사용하여 다양한 운영 체제를 실행하는 컴퓨터에서 열 수 있습니다.

텍스트 파일의 단점

- 가장 큰 단점은 서식이 없다는 것입니다. 텍스트 파일은 이미지를 포함하거나 그 의미를 전달하기 위해 디자인 요소에 의존하는 문서 (표 형식의 데이터, 수학 공식 또는 구체적인 시를 포함하는 파일)를 나타내는 데 적합하지 않습니다.
- 텍스트 파일은 순차 접근 파일이라서 앞에서 뒤로 순서대로만 접근할 수 있다. 특정 데이터 기록을 찾으려면, 프로그램이 파일의 맨 처음에서 시작하여 원하는 데이터에 도달할 때까지 순서대로 읽어 나가야 하므로, 파일의 맨 앞 에서부터 찾고 있는 데이터 기록이 있는 지점에 도달할 때까지 만큼의 시간이 소요된다.

- 순차 접근 파일은 프로그램에서 파일에 들어있는 데이터 (거의) 전부를 처리하고, 파일 내용이 거의 변경되지 않는 경우에 사용하는 것이 좋다. 이런 파일의 약점은 최신 데이터로 업데이트, 데이터 기록을 새것으로 변경, 새 기록 끼워 넣기 등이 어렵다는 점이다.

데이터를 텍스트 파일에 쓰기

델파이에서 텍스트 파일을 가지고 어떻게 작업하는지 살펴보자.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
begin
   AssignFile(f, 'my.txt');
end;
```

파일 변수 f를 my.txt라는 이름을 가진 파일에 연결하였다. 이 경우 파일의 위치는 현재 디렉토리 (실행되고 있는 프로그램이 위치한 폴더) 안이다.

이렇게 이름만 전달되는 파일 이름을 지역 파일 이름이라고 한다. 만약이 파일이 현재 디렉토리에 들어있지 않다면 이 이름 앞에 파일의 전체 경로를 붙여주어야 한다.

텍스트 파일은 읽기와 쓰기 작업을 할 수 있지만, 읽기와 쓰기를 동시에 할 수는 없다.
파일을 읽거나 쓰려면, 먼저 파일을 열어야 한다. 다음과 같이 파일을 열 수 있다.

텍스트파일 쓰기 관련 루틴

루틴 이름	기능
AssignFile(f, 'My.txt')	연결된 이름으로 빈 파일 하나를 만들어 낸다. 이런 이름을 가진 파일이 이미 있었다면, 그 파일의 내용이 모두 지워져서 열린다.
Rewrite(f)	파일 쓰기를 위해서 새 파일을 연다
Write, WriteLn	파일에 데이터를 적을 수 있다.
CloseFile(f)	파일을 닫고 쓰기를 마친다. 파일 작업이 끝나면 반드시 호출해야 한다. 호출하지 않으면, 정보 일부를 잃게 될 수도 있다.
Append(f)	새 파일을 만드는 것이 아니라 기존의 파일을 열고 그 파일의 맨 뒤에 커서를 놓는다. Append(f)는 존재하는 파일에 사용해야 한다. 그렇지 않으면, 실행하는 시스템에 오류가 표시되고 프로그램이 종료된다.

아래 예문은 'example.txt'라는 텍스트 파일에 다양한 정보를 적는 프로시저 예문이다.

```
Procedure ZapFile;
var
    f: TextFile;
    x, y: Integer;
    Ok: boolean;
begin
    AssignFile(f, 'example.txt');
    ReWrite(f);
    x:=10;
    y:=7;
    WriteLn(f, x); //파일의 첫 번째 줄에 10을 적는다, 커서는 두 번째 줄로 간다.
    Write(f, x+2); //두 번째 줄에 12 (10+2)를 적는다, 커서는 두 번째 줄 가장 뒤에 그대로 있다.
    Write(f, '안녕'); //두 번째 줄의 맨 뒤 (커서의 위치)에 "안녕" 이라는 단어가 적힌다, 커서는
두 번째 줄의 맨 뒤에 그대로 있다.
    Write(f, x, y); //커서의 위치에 10을 적고 공백이 없이 바로 7을 적는다, 커서는 두 번째 줄의 여
전히 맨 뒤에 있다.
    WriteLn(f, x, y); //커서의 위치에 10과 7을 적는다, 커서는 '세 번째'줄로 이동한다.
    Ok:=5>7;
    WriteLn (f, Ok); //(Ok 변수의 값인)False 값을 세 번째 줄에 적는다, 커서는 '네 번째'줄로
이동한다.
    WriteLn(f, x, ' ', y); //10을 적고 공백 2개를 적고 7을 네 번째 줄에 적는다, 커서는 '다섯
번째' 줄로 이동한다.
    WriteLn(f, 'x=', x); //"x=10"을 파일의 다섯 번째 줄에 적는다.
    CloseFile(f);
end;
```

아래 예문은 텍스트 파일인 **'testi.txt'**과 **'test2.txt'**를 만들고, 여기에 0을 몇 줄 적는다. 첫 번째 파일에는 0이 5개 있는 줄을 3줄 적는다. 두 번째 파일에는 0이 3개 있는 줄을 4줄 적는다. 0과 0 사이에는 공백 2개로 구분한다.

```
procedure ZapZero(var fl:  TextFile; n, m: Integer);
var
    i, j: Integer;
begin
    Rewrite(fl);
    for i:=1 to n do
    begin
        for j:=1 to m do
            Write(fl,0:3);
        WriteLn(fl)
    end;
    CloseFile(fl)
end;
procedure Tform1.Button1Click(Sender: TObject);
var
    f, g: TextFile;
begin
    AssignFile(f, 'test1.txt');
    AssignFile(g, 'test2.txt');
    ZapZero(f,3,5);
    ZapZero(g,4,3);
end;
```

텍스트 파일에서 데이터 읽기

파일에서 데이터를 읽는 절차를 살펴보자. Reset(f)을 호출하여 읽을 파일을 연다.

ReadLn 구문을 완성하고 나면, 변수 s에는 첫 번째 줄의 내용이 들어가고 커서는 그 다음 줄의 맨 앞에 놓인다. 이 ReadLn 구문을 다시 반복하면 변수 S에는 두 번째 줄의 내용이 들어가고 커서는 세 번째 줄의 맨 앞에 놓인다. 읽기 작업을 마치고 나면, 쓰기 작업을 마쳤을 때와 마찬가지로 CloseFile(f) 프로시저를 사용하여 파일을 닫는다.

아래 예문은 'my.txt 파일의 첫 번째 줄을 메모 컴포넌트인 memEx1 에 적는다. my.txt 파일을 새로 만들지 않고 앞에서 만든 파일 중 하나를 사용해도 좋다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
    f: TextFile;
    s: String;
begin
    AssignFile(f,'my.txt'); //파일 이름과 파일 변수를 연결한다.
    Reset(f); //읽을 파일을 연다.
    ReadLn(f, s); //파일의 첫 번째 줄을 읽는다.
    memEx1.Lines.Append(s); //읽은 줄을 Memo에 넣는다.
    CloseFile(f); //파일을 닫는다.
end;
```

위의 프로시저는 언제나 파일의 첫 번째 줄만 읽어서 넣는다. 파일의 여러 줄을 읽으려면 루프 반복를 사용한다. 아래 예문은 'my.txt 텍스트 파일에서 다섯 줄을 읽어서 메모에 넣는다.

텍스트파일 일기 관련 루틴

루틴 이름	기능
AssignFile(f, 'My.txt')	파일 이름과 파일 변수를 연결한다.
Reset(f)	읽을 파일을 연다. 읽으려고 하는 파일이 디스크에 있어야 한다. 그렇지 않으면, 실행 중 오류 (run time error) 메시지가 표시되고 프로그램 실행이 종료된다.
Read(f)	텍스트 파일의 경우 커서를 그 다음 줄로 이동시키지 않고 하나 이상의 값을 하나 이상의 변수로 읽습니다.
ReadLn(f,s) **ReadLn(f)**	한 줄의 텍스트를 읽고 파일의 다음 줄을 건너뜁니다. 매개 변수가 없는 ReadLn(F)는 현재 파일 위치가 다음 줄의 시작 부분으로 이동하도록 합니다.

CloseFile(f)	파일을 닫고 쓰기를 마친다. 파일 작업이 끝나면 반드시 호출해야 한다. 호출하지 않으면, 정보 일부를 잃게 될 수도 있다.
EOF(f)	EOF은 End Of File을 줄인 말로 불린 함수로 커서가 파일의 맨 뒤에 있으면 True를 반환하고, 그렇지 않으면 False를 반환한다.
EOLN(f)	커서가 줄의 맨 뒤에 있으면 True를 반환하고, 그렇지 않으면 False를 반환한다.
SeekEOL(f)	공백을 무시하고 파일의 줄 끝 상태를 반환합니다. SeekEoln을 호출하여 파일 포인터와 현재 줄의 끝 사이에 공백만 있는지 확인합니다. 공백 너머로 이동하여 현재 줄의 끝, 파일 끝 또는 공백이 아닌 다음 문자(둘 중 먼저 오는 것)에 배치합니다. 파일 포인터를 현재 줄의 끝이나 파일의 끝에 남겨두면 True를 반환합니다. 그렇지 않으면 False를 반환합니다.
SeekEOF(F)	공백을 무시하고 파일의 끝 상태를 반환합니다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   s: String;
   i: Integer;
begin
   AssignFile(f, 'my.txt'); //파일 이름과 파일 변수를 연결한다.
   Reset(f); //읽을 파일을 연다.
   for i:=1 to 5 do //5 회 반복한다.
   begin
       ReadLn(f, s); //파일에서 한 줄을 읽는다.
       memEx1.Lines.Append(s); //읽은 줄을 Memo에 넣는다.
   end;
   CloseFile(f); //파일을 닫는다.
end;
```

우리가 이미 알고 있듯이, 텍스트 파일은 순차 접근 파일이다. 즉, 앞줄을 건너뛰어서 특정 줄을 바로 들어가는 것이 불가능하다. 예를 들어 다섯 번째 줄로 들어가고 싶으면 그 앞에 있는 네 줄을 지나가야 한다. 이럴 때는 변수를 명시하지 않고 ReadLn(f)를 사용한다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
    f: TextFile;
    s: String;
    i: Integer;
begin
    AssignFile(f, 'my.txt'); //파일 이름과 파일 변수를 연결한다.
    Reset(f); //읽을 파일을 연다.
    for i:=1 to 4 do //4 회 반복한다.
        ReadLn(f); //이 줄을 건너 뛴다.
    ReadLn(f, s); //파일의 다섯 번째 줄을 읽는다.
    memEx1.Lines.Append(s); //읽은 줄을 Memo에 넣는다.
    CloseFile(f); //파일을 닫는다.
end;
```

파일의 줄 개수를 항상 미리 알 수는 없다. 그렇다면 파일의 줄 수를 모르면서 파일의 모든 줄을 읽는 것이 가능할까? 아래 예문은 Eof(f)를 사용하여 텍스트 파일에 있는 줄을 모두 읽어서 메모 컴포넌트에 넣는 프로시저이다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
    f: TextFile;
    s: String;
begin
    AssignFile(f, 'my.txt'); //파일 이름과 파일 변수를 연결한다.
    Reset(f); //읽을 파일을 연다.
    while not EOF(f) do //파일의 끝에 도달할 때까지 반복한다.
    begin
        ReadLn(f, s); //파일에서 한 줄을 읽는다.
        memEx1.Lines.Append(s); //읽은 줄을 TMemo에 넣는다.
    end;
    CloseFile(f); //파일을 닫는다.
end;
```

지금까지 모든 예문에서 Read가 아니라 ReadLn 구문을 사용했다는 점을 눈여겨보자. 마지막 예문에서 ReadLn을 Read로 바꾸게 되면 읽기 작업이 끝나지 않는다. 왜냐하면 Read는 커서를 그 다음 줄로 이동시키지 않기 때문에 커서가 그 줄의 맨 뒤에 그대로 남아있게 되고 Read는 빈 줄을 읽게 된다. 따라서 끝나지 않는 루프에 빠져서 빈 줄을 끝없이 읽어서 Memo에 넣을 것이다. 커서가 결코 파일의 끝에 도달하지 못하기 때문이다. 따라서 파일에서 줄을 읽을 때에는 ReadLn 구문을 사용할 필요가 있다.

이번에는 파일에서 실수나 정수를 읽어보자.
이 숫자들이 공백으로 구분되어 있으면 된다. 이 경우 숫자를 직접 읽을 수 있다. 어떻게 하는지 예문을 살펴보자.

텍스트 파일에 숫자들이 들어있고, 이 숫자 사이에는 하나 이상의 공백이 있다고 가정한다. 예문은 아래 그림과 같다(공백임을 알 수 있게 여기에서는 물결 표시를 하였다).

파일을 열고 나서, Read(f, x) 구문을 사용한다. f는 파일 변수이고 x는 Integer 변수이다. 이 구문이 실행되고 나면, x 변수에는 10이 들어가고, 커서는 0자 바로 뒤에 있는 구분자(여기 서는 공백)앞에 있다. Read(f, x)가 두 번째 실행되고 나면, 커서는 모든 구분자(공백)을 건 너 뛰고 나서 7이 변수 x에 들어간다. 커서는 다시 7 뒤에 있는 공백에서 멈춘다. Read(f, x) 가 세 번째 실행되고 나면, 숫자 25 앞에 있는 모든 구분자(공백)을 건너 뛰고 25를 읽어서 변수 x에 넣는다. 말하자면, Integer 변수인 x를 사용하는 Read(f, x)는 그 숫자 앞에 있는 모든 구분자를 건너 뛰어서 그 숫자를 읽은 후에 숫자가 아닌 기호(또는 문자) 앞에 커서를 놓는다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
    f: TextFile;
    x, i, k: Integer;
```

```
begin
    AssignFile(f,'my.txt'); //파일 이름과 파일 변수를 연결한다
    Reset(f); //읽을 파일을 연다
    k:=0;
    for i:=1 to 5 do //5 회 실행한다:
    begin
        Read(f, x); //다음 숫자를 읽고 나서
        k:=k+x; //읽은 숫자를 합계에 더한다
    end;
    CloseFile(f); //파일을 닫는다.
    ShowMessage(IntToStr(k)); //합계를 표시한다
end;
```

위의 프로시저가 실행되고 나면, 변수 k에는 57이 들어간다 57은 앞에 있는 숫자 5개 즉 10, 7, 25, 14, 1의 합계이다. Read(f, x)는 숫자 데이터를 읽어서 Integer변수에 넣는다. 구분자 기호들은

모두 건너뛴다. Read에서 구분자는 공백과 줄-끝남(end-of-line) 기호이다.

위 예문에서, Read(f, x) 구문을 ReadLn(f, x)로 바꾸면 어떻게 될까?
ReadLn이 Read와 가장 다른 점은 ReadLn은 커서를 다음 줄에 보내야 하기 때문에 나머지 데이터를 모두 건너 뛴다는 점이다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
f: TextFile;
x, i, k: Integer;
begin
    AssignFile(f,'my.txt'); //파일 이름과 파일 변수를 연결한다.
    Reset(f); //읽을 파일을 연다.
    k:=0;
    for i:=1 to 5 do //5 회 실행한다.
    begin
        ReadLn(f, x); //다음 숫자를 읽고 나서
        k:=k+x; //읽은 숫자를 합계에 더한다.
    end;
    CloseFile(f); //파일을 닫는다.
    ShowMessage(IntToStr(k)); //합계를 표시한다.
end;
```

위 프로시저가 실행되고 나면, 변수 k 에는 86이 들어간다. 86은 앞에 있는 5 줄에서 가장 앞에 있는 숫자들 (즉 10, 14, 24, 27, 11)의 합이다.
텍스트 파일 안에 있는 모든 숫자를 읽어야 하는 상황이 종종 있다. 게다가 전체 숫자의 개 수를 미리 알 수가 없다. 이 경우 EOLN(f) 함수를 사용하여 줄이 끝났는지는 상황을 체크하면서 숫자 읽기 구문을 반복한다. 이 경우에는 프로시저가 다음과 같다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
    f: TextFile;
    x, k: Integer;
begin
    AssignFile(f,'my.txt'); //파일 이름과 파일 변수를 연결한다.
    Reset(f); //읽을 파일을 연다.
    k:=0;
    while not EOLN(f) do //줄 끝에 도달하기 전까지 반복
    begin
        Read(f, x); //다음 숫자를 읽는다.
        k:= k + x; //읽은 숫자를 합계에 더한다.
    end;
    CloseFile(f); //파일을 닫는다.
    ShowMessage(IntToStr(k)); //합계를 표시한다.
end;
```

위 예문이 실행되고 나면, 변수 k에는 42가 들어간다. 42는 첫 번째 줄에 있는 모든 숫자 즉, 10, 7, 25의 합계이다. 하지만, 하지만 반드시 42인 것은 아니다. 읽은 줄에 있는 마지 막 숫자 뒤에 공

백이 있는지 없는지에 따라 달라지기 때문이다. 공백이 없다면, 42가 될 것 이다.

위 상황에서 만약 숫자 25 뒤에 공백이 있다면 어떻게 될까? 25를 읽은 다음, 커서는 숫자 5의 바로 뒤에 놓인다. EOLN(f)를 호출하면, False가 반환되는 데 그 공백이 줄 끝남 기호
가 아니기 때문이다. 이 루프는 한 번 더 실행될 것이고 Read(f, x)는 모든 구분자를 건너 뛰게 된다. 구분자인 줄 끝남 기호도 구분자이므로 역시 건너 뛰고 다음 숫자가 나올 때 까지 간다. 그 결과, k 변수에는 몇 개의 줄에서 읽어 낸 숫자들이 더해질 수 있다. 줄의 개수 읽은 숫자의 개수는 줄 끝남 앞에 공백을 가진 줄이 몇 개나 이어져 있는가에 따른다.

모든 공백을 건너 뛰고 나서 줄-끝남 기호를 확인한다. 위 예문에서 EOLN(F) 대신 SeekEOL 이 문제를 피하려면, 숫자를 읽을 때 EOLN(f) 대신 SeekEOLN(f) 함수를 사용해야 한다. SeekEOLN(f)은 먼저 EOLN(f)를 호출하면 줄 끝남 기호 앞에 얼마든지 공백이 있어도 올바르게 작동할 것이다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   x, k: Integer;
begin
   AssignFile(f,'my.txt'); //파일 이름과 파일 변수를 연결한다.
   Reset(f); //읽을 파일을 연다.
   k:=0;
   while  not  SeekEOLN(f) do //줄 끝에 도달하기 전까지 반복
   begin
       Read(f, x); //다음 숫자를 읽는다.
       k:=k+x; //읽은 숫자를 합계에 더한다.
   end;
   CloseFile(f); //파일을 닫는다.
   ShowMessage(IntToStr(k)); //합계를 표시한다.
end;
```

델파이에는 SeekEOF(F) 함수도 있다. SeekEOF(F) 함수는 SeekEOLN(F)과 같은 방식으로 작동된다. 하지만 공백과 줄 끝 기호를 모두 건너 뛴다. Read(f, x) 구문을 이용하여 파일에서 숫자를 읽을 때 사용하면 좋다.

파일에 들어있는 모든 숫자의 합계를 계산하는 프로시저는 다음과 같다.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
var
   f: TextFile;
   x, k: Integer;
begin
   AssignFile(f,'my.txt'); //파일 이름과 파일 변수를 연결한다.
   Reset(f); //읽을 파일을 연다.
   k:=0;
   while not SeekEOF(f) do //파일 끝에 도달하기 전까지 반복
   begin
      Read(f, x); //다음 숫자를 읽는다.
      k:=k+x; //읽은 숫자를 합계에 더한다.
   end;
   CloseFile(f); //파일을 닫는다.
   ShowMessage(IntToStr(k)); //합계를 표시한다.
end;
```

텍스트 파일에서 데이터를 읽는 프로그램을 작성하는 지침으로 다음 규칙을 기억하자.

- 데이터를 줄로 읽으려면, EOLN(f) 과 EOF(f) 함수를 사용한다.
- 데이터를 숫자로 읽으려면 SeekEOLN(f) 과 SeekEOF(f) 함수를 사용한다.

프로그램이 파일에서 숫자가 아닌 데이터(즉, 파일 끝남 기호나 문자로 된 글)를 읽으려고 하면, 실행 중 시스템이 오류 메시지를 표시하고 프로그램은 실행이 종료된다. 프로그램이 실수를 읽어서 정수 변수에 넣으려고 하거나, 델파이의 숫자 형식이 아닌 (소수 부분을 마침 표가 아니라 쉼표로 구분하는 등) 숫자를 읽으려고 하는 경우에도 역시 오류 메시지를 표시하고 실행이 종료된다

[연습 문제]

1. 파일 안에서 가장 짧은 줄을 찾아 내고 그 내용을 레이블 컴포넌트에 표시하자. 만약 길이 가 같은 가장 짧은 줄이 몇 개 있으면, 그 중 가장 마지막에 찾아낸 줄의 내용을 표시하자. 파일의 이름은 텍스트박스에서 입력 받는다. 텍스트 파일을 미리 만들어 두고 사용하자.
2. 텍스트 파일에 몇 줄이 들어 있다. 각 줄에는 정수들이 몇 개씩 있다. 이 정수들은 하나 이상의 공백에 의해 떨어져있다. 파일을 읽어서 줄에 있는 숫자의 합계가 짝수인 줄을 다른 파일에 옮겨 적어보자. 파일 이름은 텍스트박스에서 입력 받는다(OpenDialog를 사용해도 된다).

프로그램이 잘 작동하는 지 테스트하기 위해 메모장등 외부 텍스트 편집기를 사용 하여 텍스트 파일을 미리 만들어 두고 사용하자