오브젝트 파스칼로 프로그래밍 시작하기

한국어판

글쓴이: Motaz Abdel Azeem
편집자: Pat Anderson, Jason Hackney
번역, 조판: 조성호 (darkcircle.0426@gmail.com)
그림: 흡혈양파 (http://www.onionmixer.net/)
감수: 백록화

code.sd

2011년 6월 18일
번역: 2012년 3월 5일
LaTeX조판: 2012년 8월 16일

도입

오브젝트 파스칼 언어를 배우고 싶어하는 프로그래머를 위해 이 책을 썼습니다. 신입생과 프로그래머가 아닌 분들을 위한 프로그래밍 입문서로도 안성맞춤입니다. 오브젝트 파스칼 언어에 대한 내용에 덧붙여 일반적으로 사용하는 프로그래밍 기술에 대해 설명했습니다.

오브젝트 파스칼 언어

객체지향 프로그래밍을 지원하는 파스칼 언어는 1983년 애플 컴퓨터사에서 처음 선보였습니다.이후 볼랜드사에서 그들의 유명한 터보 파스칼 계열에 객체지향 프로그래밍을 지원했습니다. 오브젝트 파스칼은 범용 하이브리드 (구조화된 객체지향 프로그래밍) 언어입니다. 이 언어는 학습, 게임 개발, 사업용 프로그램, 인터넷 프로그램, 통신 프로그램, 도구 개발, OS 커널과 같은 다양한 범주의 프로그램을 만들기 위해 사용할 수 있습니다.

델파이

터보 파스칼이 성공하자, 볼랜드는 윈도우즈 환경에 이식하기로 결정했으며, 구성요소 기반 기술을 이것에 도입했습니다. 곧 델파이는 그 당시 최고의 RAD (Rapid Application Development) 도구가 되었습니다. 델파이의 첫번째 버전은 윈도우즈와 데이터베이스 프로그램 개발을 지원하는 풍부한 구성요소와 패키지 모음과 함께 1995년에 출시했습니다.

프리 파스칼

볼랜드사가 터보 파스칼 계열의 지원을 중단한 후, 프리 파스칼 팀은 터보 파스칼과 호환되는 컴파일러를 바닥부터 완전히 새로 작성하기 시작했고 델파이와 호환되도록 했습니다. 이때 프리 파스칼 컴파일러는 윈도우즈, 리눅스, 맥 그리고 WinCE와 같은 부가적인 플랫폼을 대상으로 삼았습니다.

프리 파스칼 컴파일러 버전 1.0은 2000년 6월에 출시했습니다.

라자루스

프리 파스칼은 컴파일러이며, 윈도우즈용 델파이 IDE와 같은 통합 개발환경(IDE)이 빠져 있습니다. 라자루스 프로젝트는 프리 파스칼을 위한 IDE를 제공하기 위해 시작했습니다. 이는 코드 편집기, 디버거를 제공하고, 델파이 IDE와 유사하게 수많은 프레임워크, 패키지 그리고 구성요소 라이브러리를 포함합니다.

라자루스 버전 1.0은 아직 (정식으로) 출시되지 않았습니다만 라자루스 베타 버전으로 개발된 수많은 프로그램들이 있습니다. 많은 공헌자 분들께서 라자루스를 위한 패키지와 구성요소를 작성하며, 이 커뮤니티는 계속 성장해 나가고 있습니다.

오브젝트 파스칼의 특징

오브젝트 파스칼은 매우 쉽고, 초보자들도 쉽게 알아볼 수 있는 언어이며, 컴파일러는 매우 빠르고, 이 컴파일러가 만들어낸 프로그램은 빠르고 신뢰성 있으며, C와 C++

에 견주어 볼 수 있습니다. 여러분은 이 IDE (라자루스와 델파이) 를 통해 견고하고 거대한 프로그램을 복잡하지 않게 작성할 수 있습니다.

저자: Motaz Abdel Azeem

저는 1999년 수단 과학기술 대학을 졸업했습니다. 그리고 BASIC을 배운 후에 두번째 언어로 파스칼을 배우기 시작해습니다. 이후에는 이것을 줄곧 사용해왔고, C와 C++을 배운 후에 특히 이것이 매우 쉽고 강력한 도구임을 깨달았습니다. 이후 전 델파이로 전향했습니다. 전 제 모든 프로그램을 작성하기 위해 델파이와 라자루스를 사용했습니다. 카르툼에 살고 있습니다. 제 현재 직업은 소프트웨어 개발자 입니다.

첫번째 편집자

Pat Anderson은 1968년 서부 워싱턴 주립 대학을 졸업했고 1975년 러트거스 로스쿨을 졸업했습니다. 그는 워싱턴주 스노퀄미에서 시 변호사로 일하고 있습니다. Pat은 1982년 Radio Shack TRS 80 Model III의 내장 BASIC 인터프리터를 사용하여 프로그래밍을 시작했지만 곧 터보파스칼을 발견했습니다. 그는 터보 파스칼 4.0부터 7.0까지, 그리고 델파이 1.0부터 4.0까지의 모든 버전을 모든 버전을 소유하고 있습니다. Pat은 1998년부터 그의 프로그래밍에 대한 열정이 다시 불타오르게 해준 프리 파스칼 / 라자루스를 통해 돌아온 2009년 전까지 프로그래밍을 중단했습니다.

두번째 편집자

Jason Hackney는 서부 미시간 대학 항공학과를 졸업했습니다. 그는 남동부 미시간 지역을 기반으로 하는 발전시설 회사의 정규직 전문 비행사로 일을 하고 있습니다. Jason은 1984년경 Commodore 64를 통해 데뷔한 평범한 프로그래머였습니다. 터보 파스칼이 대략적으로 도입된 1990년, 그는 리눅스, 라자루스 그리고 프리 파스칼을 발견한 이후로 숨어있던 프로그래밍에 대한 흥미가 다시 살아났습니다.

번역

저는 순천향대학교 정보기술공학부를 졸업했습니다. 고려대학교 전자전기전파공학부 석사 과정중에 있으며, GNOME Korea 로컬 팀과 Xfce 데스크탑 환경 한국 로컬 커미터로 활동하고 있습니다. 오브젝트 파스칼에 대해서는 델파이와 카일릭스 같은 개발도구의 이름만 들어본 수준이었을뿐 전혀 몰랐습니다.
흡혈양파(http://www.onionmixer.net/)님께서 이 책을 소개해주시면서 이 언어에 관심을 갖게 되었고, 원서의 번역에 참여하게 되었습니다. 이 책을 소개해주신 흡혈양파님, 그리고 제가 보잘 것 없는 실력으로 번역한 이 번역서를 흔쾌히 감수해주신 백록화님께 깊은 감사의 말씀을 드립니다.

라이선스

이 책의 예제를 위한 환경

이 책의 모든 예제를 위해 라자루스와 프리 파스칼을 사용할 것입니다. 여러분은 프리 파스칼 컴파일러가 들어있는 라자루스 IDE를 다음 사이트에서 받으실 수 있습니다: http://lazarus.freepascal.org

리눅스를 사용하고 있다면 프로그램 저장소에서 라자루스를 가져올 수 있습니다. 우분투에서는 다음 명령을 사용할 수 있습니다.

```
sudo apt-get install lazarus
```

페도라에서는 다음 명령을 사용할 수 있습니다.

```
yum install lazarus
```

라자루스는 자유 오픈소스 프로그램입니다. 그리고 다양한 플랫폼에서 사용할 수 있습니다.

라자루스에서 작성한 프로그램은 다른 플랫폼에서 실행할 수 있도록 해당 플랫폼에서 재컴파일 할 수 있습니다. 예를 들면, 여러분이 윈도우즈에서 라자루스를 이용하여 프로그램을 작성하고 리눅스에서 해당 프로그램을 실행가능하게 만들고 싶다면, 단지 리눅스에 설치된 라자루스에 소스 코드를 복사하고, 컴파일 하시면 됩니다.

라자루스는 각각의 운영체제에서 동작하는 고유한 프로그램을 만들며, 어떤 추가적인 라이브러리나 가상 머신이 필요하지 않습니다. 이러한 이유로 배포하기 쉽고, 실행이 빠릅니다.

텍스트 모드 사용하기

이 책의 첫번째 장의 모든 예제는 콘솔 프로그램(텍스트 모드 프로그램 / 명령 줄 프로그램)일 것입니다. 왜냐면, 그 프로그램들은 이해하기 쉬우며 표준이기 때문입니다. 그래픽 사용자 인터페이스 프로그램은 그 다음 장에서 소개될 것입니다.

목 차

Chapter 1

언어기초

1.1 우리의 첫번째 프로그램

라자루스를 설치하고 실행하고 나면, 메인메뉴에서 새로운 프로그램의 작성을 시작할 수 있습니다.

/ Project / New Project / Program

소스 코드 창에서 다음 코드를 보게 될 것입니다.

```
Program Project1;
{$mode objfpc}{$H+}
uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

{$IFDEF WINDOWS}{$R project1.rc}{$ENDIF}
begin
end.
```

메인 메뉴에서 File / Save를 눌러 이 프로그램을 저장할 수 있으며, 이에 대해 first.lpi와 같은 이름을 지정할 수 있습니다.

그러고 나면 *begin*과 *end* 구문 사이에 이 줄들을 작성할 수 있습니다.

```
    Writeln('This is Free Pascal and Lazarus');
    Writeln('Press enter key to close');
    Readln;
```

완성된 소스 코드는 다음과 같을 것입니다.

```
Program Project1;
{$mode objfpc}{$H+}
uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

{$IFDEF WINDOWS}{$R project1.rc}{$ENDIF}
begin
    Writeln('This is Free Pascal and Lazarus');
    Writeln('Press enter key to close');
    Readln;
end.
```

Writeln 구문은 텍스트를 스크린(콘솔 창)에 표시합니다. Readln 구문은 여러분이 프로그램을 닫기 위해 엔터를 누르기 전까지 사용자 여러분이 화면에 표시한 텍스트를 읽어들이는 것을 실행하고 라자루스 IDE로 복귀합니다.

이제 프로그램을 실행하기 위해 *F9*를 누르거나 다음 버튼을 눌러봅니다.

첫 프로그램을 실행하면, 다음 출력 텍스트를 보게 될 것입니다.

```
This is Free Pascal and Lazarus
Press enter key to close
```

리눅스를 사용하고 있다면 프로그램 디렉터리에서 (first) 라고 하는 새 파일을 찾게 될 것이고, 윈도우즈에서는 first.exe로 이름지은 파일을 찾게 될 것입니다. 두 파일 다 마우스로 두 번 두르면 바로 실행할 수 있습니다. 실행 가능한 파일은 라자루스 IDE 없이 실행하도록 다른 컴퓨터에 복사할 수 있습니다.

참고

콘솔 프로그램 창이 뜨지 않는다면 라자루스 메뉴에서 디버거를 비활성화 할 수 있습니다.

Environment / Options / Debugger

Debugger type and path에서 (none)을 선택합니다.

1.2 또 다른 예제

이전 프로그램에서 다음 줄을

```
Writeln('This is Free Pascal and Lazarus');
```

다음과 같이 바꿔봅니다.

```
Writeln('This is a number:', 15);
```

다 되었다면 *F9*키를 눌러 프로그램을 실행해봅니다.

여러분은 다음의 결과를 보게 될 것입니다.

```
This is a number: 15
```

앞에 보여드린 줄을 다음과 같이 바꿔보고, 제각각의 경우에 대해 프로그램을 실행해봅니다.

코드:

```
Writeln('This is a number:', 3 + 2);
```

출력:

```
This is a number: 5
```

코드:

```
Writeln('5 * 2 = ', 5 * 2);
```

출력:

```
5 * 2 = 10
```

코드:

```
    Writeln('This is real number:', 7.2);
```

출력:

```
This is real number: 7.2000000000000E+0000
```

코드:

```
    Writeln('One, Two, Three:', 1, 2, 3);
```

출력:

```
One, Two, Three: 123
```

코드:

```
    Writeln( 10, '*', 3,'=', 10 * 3);
```

출력:

```
10 * 3 = 30
```

제각각의 경우를 통해 Writeln 구문에 다른 값들을 집어넣고 그에 대한 결과를 볼 수 있습니다. 이 소스 코드들은 우리가 완전히 이해할 수 있게 도움을 줄 것입니다.

1.3 변수

변수는 데이터 저장소입니다. 예를 들어, 우리가 X = 5라고 했다면, X는 변수이며, 여기에 값 5가 들어있음을 의미합니다.

오브젝트 파스칼은 강타입 언어이며, 이는 값을 변수에 넣기 전에 변수의 형을 선언해야 함을 의미합니다. X를 정수형으로 선언했다면, 프로그램이 실행하는 동안 X에 정수형 숫자만 넣어야 합니다.

다음은 변수를 선언하고 사용하는 예제입니다.

```
Program FirstVar;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    {you can add units after this};

{$IFDEF WINDOWS}{$R project1.rc}{$ENDIF}
var
    x: Integer; begin
    x:= 5;
    Writeln(x * 2);
    Writeln('Press enter key to close');
    Readln;
end.
```

우리는 프로그램의 출력에서 10을 볼 수 있을 것입니다.

참고로, 다음 줄이 선언문이 됨을 의미하는 *var*라는 예약어를 사용했습니다.

```
    x: Integer;
```

이는 다음 두 가지를 의미합니다.

1. 변수의 이름은 X이고

2. 이 변수의 형은 분수 부분을 제외한 정수 숫자값만 지닐 수 있는 Integer 입니다. 양수 값처럼 음수 값도 지닐 수 있습니다.

그리고 다음 구문은

```
    x:= 5;
```

값 5를 변수 x에 넣음을 의미합니다.

다음 예제에서 변수 y를 추가해보았습니다.

```
var
    x, y: Integer;
begin
    x:= 5;
    y:= 10;
    Writeln(x * y);
    Writeln('Press enter key to close');
    Readln;
end.
```

위의 프로그램에 대한 출력은

```
50
Press enter key to close
```

50은 식 (x*y)의 결과입니다.

다음 예제를 통해 *character*라는 새로운 데이터 형을 소개합니다.

```
var
    c: Char;
begin
    Writeln('My first letter is :', c);
    Writeln('Press enter key to close');
    Readln;
end.
```

이 형은 값과 같은 것이 아닌 단 하나의 문자나 문자 형태의 숫자를 지닐 수 있습니다.

다음 예제에서는 분수 부분을 지닐 수 있는 *real* 숫자 형을 소개합니다.

```
var
    x: Single;
begin
    x:= 1.8;
    Writeln('My car engine capacity is ', x, 'liters');
    Writeln('Press enter key to close');
    Readln;
end.
```

좀 더 상호적인 유연한 프로그램을 작성하려면 사용자로부터 입력을 받아들일 필요가 있습니다. 예를 들어, 사용자에게 숫자를 입력해달라고 요청하고, Readln 구문 / 프로시저를 통해 사용자의 입력으로부터 숫자를 가져올 수 있습니다.

```
var
    x: Integer;
begin
    Write('Please input any number');
    Readln(x);
    Writeln('You have entered: ', x);
    Writeln('Press enter key to close');
    Readln;
end.
```

이 예제에서는, 프로그램에서 상수 값을 받아들이는 대신에 키보드를 통해 X에 값을 할당했습니다.

아래 예제에서는 사용자가 입력한 값에 대한 곱셈표를 보여줍니다.

```
Program MultTable;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

var
    x: Integer;
begin
    Write('Please input any number:');
    Readln(x);
    Writeln(x,' * 1 = ',x * 1);
    Writeln(x,' * 2 = ',x * 2);
    Writeln(x,' * 3 = ',x * 3);
    Writeln(x,' * 4 = ',x * 4);
    Writeln(x,' * 5 = ',x * 5);
    Writeln(x,' * 6 = ',x * 6);
    Writeln(x,' * 7 = ',x * 7);
    Writeln(x,' * 8 = ',x * 8);
    Writeln(x,' * 9 = ',x * 9);
    Writeln(x,' * 10 = ',x * 10);
    Writeln(x,' * 11 = ',x * 11);
    Writeln(x,' * 12 = ',x * 12);
    Writeln('Press enter key to close');
    Readln;
end.
```

참고로 앞의 예제에서 작은 따옴표(') 사이에 있는 모든 텍스트는 콘솔 윈도우에 그대로 보여줍니다. 예를 들면 다음과 같습니다.

```
' * 1 = '
```

작은 따옴표에 둘러싸이지 않은 변수와 수식은 계산되며 값으로 기록합니다.

다음 두 구문의 차이점을 보도록 합니다.

```
    Writeln('5 * 3');
    Writeln(5 * 3);
```

첫번째 구문의 결과는 다음과 같습니다.

```
5 * 3
```

두번째 구문은 계산된 다음의 결과를 표시합니다.

```
15
```

다음 예제에서는 두 숫자들(x, y)에 대한 수학적인 계산을 수행하고, 이에 대한 결과를 세번째 변수(Res)에 넣을 것입니다.

```
var
    x, y: Integer;
    Res: Single;
begin
    Write('input a number:');
    Readln(x);
    Write('input another number:');
    Readln(y);
    Res:= x / y;
    Writeln(x,'/',y,'=',Res);
    Writeln('Press enter key to close');
    Readln;
end.
```

나누기 연산을 수행하면, 분수가 붙은 숫자 결과가 나올 수 있기 때문에 결과 변수(Res)를 실수(*Single*)로 선언했습니다. *Single*은 단순 정밀도 실수를 의미합니다.

1.4 하위 형식

변수에 대해 여러가지 하위 형식이 존재하는데, 예를 들자면, 정수 숫자의 하위 형식은 값의 범위와 메모리에 값을 저장하는데 필요한 바이트 수에 차이를 두고 있습니다.

다음 표에는 정수 형, 값의 범위, 그리고 메모리에 필요한 바이트 수가 있습니다.

형	최소값	최대값	바이트크기
Byte	0	255	1
ShortInt	-128	127	1
SmallInt	-32768	32767	2
Word	0	65535	2
Integer	-2147483648	2147483647	4
LongInt	-2147483648	2147483647	4
Cardinal	0	4294967295	4
Int64	-9223372036854780000	9223372036854775807	8

아래 예제와 같이, Low, High 그리고 SizeOf 함수를 사용하여, 각각의 함수들로부터 제각각의 하위 형식에 대한 최소, 최대값과 바이트 크기를 가져올 수 있습니다.

```
Program Types;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes

begin

    Writeln('Byte: Size = ', SizeOf(Byte),
    ', Mininum value = ', Low(Byte), 'Maximum value = ',
    High(Byte));

    Writeln('Integer: Size = ', SizeOf(Integer),
    ', Mininum value = ', Low(Integer), 'Maximum value = ',
    High(Integer));
    Write('Press enter key to close');
    Readln;
end.
```

1.5 상태 분기

지능적 장치(컴퓨터나 프로그래밍 가능한 장치와 같은 것)들에게 있어 가장 중요한 것 중 하나는 제각기 다른 상태에 대해 동작을 취하는 것입니다. 이것은 상태 분기를 통해 해결할 수 있습니다. 예를 들어 어떤 차가 속도를 40Km/h 초과하면 문을 닫는다고 합시다. 이 경우의 상태는 다음과 같습니다.

if Speed is >= 40 and doors are unlocked, then lock door.

자동차, 세탁기, 그리고 다른 수많은 장치들은 마이크로 컨트롤러나 ARM같은 작은 크기의 프로세서처럼 프로그래밍 가능한 회로를 지니고 있습니다. 이러한 회로는 구조에 따라 어셈블리나 C, 프리 파스칼을 사용한 프로그램으로 구성할 수 있습니다.

1.6 if 조건문

파스칼 언어에서 if 조건문은 매우 간단하고 명확합니다. 아래 예를 통해, 우리는 방안의 온도에 따라 에어콘을 켤지 말지에 대해 결정하고자 합니다.

1.7 에어콘 프로그램

```
var
    Temp: Single;
begin

    Write('Please enter Temperature of this room :');
    Readln(Temp);

    if Temp > 22 then
        Writeln('Please turn on air-condition');
    else
        Writeln('Please turn off air-condition');

    Write('Press enter key to close');
    Readln;
end.
```

if then else 구문과 이에 대한 예제를 소개했습니다. 온도가 22보다 크면, 처음 문장을 보여줍니다

```
'Please turn on air-condition'
```

조건을 만나지 못했다면 (22보다 작거나 같다면), 다음 줄을 보여줍니다.

```
'Please turn off air-condition'
```

여러 상태에 대해 다음과 같이 작성할 수 있습니다.

```
var
    Temp: Single;
begin

    Write('Please enter Temperature of this room :');
    Readln(Temp);

    if Temp > 22 then
        Writeln('Please turn on air-condition');
    else
    if Temp < 18 then
        Writeln('Please turn off air-condition');
    else
        Writeln('Do nothing');

    Write('Press enter key to close');
    Readln;
end.
```

결과를 보기 위해 다른 온도 값으로 위 예제를 시험해 볼 수 있습니다.

더욱 쓸모있게 만들기 위해 조건을 더욱 복잡하게 만들 수 있습니다.

```
var
    Temp: Single;
    ACIsOn: Byte;
begin

    Write('Please enter Temperature of this room :');
    Readln(Temp);
    Write('Is air conditioner on? if it is (On) write 1,',
    ' if it is (Off) write 0 : ');
    Readln(ACIsOn);

    if (ACIsOn = 1) and ( Temp > 22 ) then
        Writeln('Do nothing, we still need cooling')
    else
    if (ACIsOn = 1) and ( Temp < 18 ) then
        Writeln('Please turn off air-conditioner')
    else
    if (ACIsOn = 0) and ( Temp < 18 ) then
        Writeln('Do nothing, it is still cold')
    else
    if (ACIsOn = 0) and ( Temp > 22 ) then
        Writeln('Please turn on air-conditioner')
    else
        Writeln('Please enter a valid values');

    Write('Press enter key to close');
    Readln;
end.
```

위 예제에서, 첫번째 상태가 *True*를 되돌리고(ACIsOn = 1), 두번째 상태가 *True*를 되돌릴 때(Temp > 22), *Writeln* 구문을 실행하라는 의미의 새로운 키워드(**and**)를 사용했습니다. 하나 혹은 양 쪽의 상태가 *False*라면, else 부분으로 진행할 것입니다.

예를 들어, 에어콘을 시리얼 포트를 통해 컴퓨터에 연결했다면, 시리얼 포트 프로시저/컴포넌트를 이용하여 프로그램에서 에어콘을 켜고 끌 수 있습니다. 이 경우 에어콘이 얼마나 오랫동안 동작하고 있었는지와 같은 if 조건문에 대한 추가 인자가 필요합니다. 만약 허용 시간(예를 들어 1시간)을 초과했다면 방 온도와는 상관 없이 에어콘을 꺼야 할 것입니다. 또한 냉기 손실율을 고려할 수 있는데, (밤에) 더워지는 속도가 매우 느리다면, 오랜 시간동안 끌 수 있을 것입니다.

1.8 체중계 프로그램

이 예제를 통해 사용자의 키를 미터 단위로 입력 받고, 몸무게를 킬로 단위로 입력해 달라고 요청할 것입니다. 그 다음 프로그램은 사용자가 입력한 값에 따라 사용자의 적당한 몸무게를 계산할 것이고, 결과를 알려줄 것입니다.

```
Program Weight;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes

var
    Height: Double;
    Weight: Double;
    IdealWeight: Double;
begin

    Write('What is your height in meters (e.g. 1.8 meter) : ');
    Readln(Height);
    Write('What is your weight in kilos : ');
    Readln(Weight);

    if Height >= 1.4 then
        IdealWeight:= (Height - 1) * 100
    else
        IdealWeight:= Height * 20;
    if (Height < 0.4) or ( Height > 2.5) or ( Weight < 3) or
        (Weight > 200) then
    begin
        Writeln('Invalid values');
        Writeln('Please enter proper values');
    end
    else
    if IdealWeight = Weight then
        Writeln('Your weight is suitable')
    else
    if IdealWeight > Weight then
        Writeln('You are under weight, you need more ',
        Format('%.2f', [IdealWeight - Weight]), ' Kilos')
    else
        Writeln('You are over weight, you need to lose ',
        Format('%.2f', [Weight - IdealWeight]), ' Kilos');
    Write('Press enter key to close');
    Readln;
end.
```

이 예제에서 새로운 키워드를 사용했습니다.

1. ***Double*** : *Single*과 유사합니다. 둘 다 실수지만, Double은 배 정밀도 부동 소숫점이며, Single 은 4바이트의 메모리 공간이 필요하지만, Double은 8바이트의 메모리 공간이 필요 합니다.

2. 두번째로 새로운 것은 키워드(***Or***)이며, 조건들 중 하나를 만났는지 확인하기 위해 사용했습니다. 조건들 중 하나를 만났다면, 구문을 실행할 것입니다. 예를 들어보겠습니다: 첫번째 조건이 *True*를 되돌렸다면 (Height $<$ 0.4), *Writeln* 구문을 호출할 것입니다:
Writeln('Invalid values');. 첫번째 조건이 *False*를 되돌렸다면 두번째 조건을 검사할 것이며 나머지의 경우에도 마찬가지 입니다. 만약 모든 조건이 *False*를 되돌렸다면, *else* 부분으로 진행할 것입니다.

3. if 조건문에 ***begin end*** 키워드를 사용한 이유는 *if* 구문이 하나의 구문을 실행하기 때문입니다. *begin end*는 여러 개의 구문들을 하나의 블럭(구문)으로 간주하는 것으로 바꿔주고, 여러개의 구문들은 *if* 구문에 의해 실행될 수 있습니다. 다음 두 개의 구문을 보도록 합니다.

```
Writeln('Invalid values');
Writeln('Please enter proper values');
```

이들은 *begin end*를 사용하여 하나의 구문으로 바뀌었습니다.

```
if (Height < 0.4) or ( Height > 2.5) or ( Weight < 3) or
    (Weight > 200) then
begin
    Writeln('Invalid values');
    Writeln('Please enter proper values');
end
```

4. 정해진 형식으로 값을 표시하는 *Format* 프로시저를 사용했습니다. 이 경우 우리는 소숫점 아래 두 자리만 표시하려고 합니다. 이 함수를 사용하기 위해 *SysUtils* Unit을 *Uses* 절에 추가할 필요가 있습니다.

```
What is your height in meters (e.g. 1.8 meter) : 1.8
What is your weight in kilos : 60.2
You are under weight, you need more 19.80 Kilos
```

참고

이 예제가 100% 정확한 것은 아닙니다. 웹에서 몸무게를 계산하는 방법을 자세하게 찾아볼 수 있습니다. 단지 프로그래머가 각각의 문제를 어떻게 해결해 나가는지 설명하고, 믿을 수 있는 프로그램을 만들어나가면서 과제에 대한 분석을 우수하게 수행하는데만 의미를 두었습니다.

1.9 case .. of 구문

상태 분기에 대한 또 다른 방법이 있는데, 그것이 바로 *case .. of* 구문 입니다. *case* 순서 값에 따라 실행을 나눕니다. 음식점 프로그램은 *case of* 구문의 사용 방법을 보여줄 것입니다.

1.10 음식점 프로그램

```
var
    Meal: Byte;
begin
    Writeln('Welcome to Pascal Restaurant. Please select your order');
    Writeln('1 – Chicken       (10$)');
    Writeln('2 – Fish          (7$)');
    Writeln('3 – Meat          (8$)');
    Writeln('4 – Salad         (2$)');
    Writeln('5 - Orange Juice (1$)');
    Writeln('6 – Milk          (1$)');
    Writeln;
    Write('Please enter your selection: ');
    Readln(Meal);
    case Meal of
    1: Writeln('You have ordered Chicken,',
      ' this will take 15 minutes');
    2: Writeln('You have ordered Fish, this will take 12 minutes');
    3: Writeln('You have ordered meat, this will take 18 minutes');
    4: Writeln('You have ordered Salad, this will take 5 minutes');
    5: Writeln('You have ordered Orange juice,',
      ' this will take 2 minutes');
    6: Writeln('You have ordered Milk, this will take 1 minute');
    else
        Writeln('Wrong entry');
    end;

    Write('Press enter key to close');
    Readln;
end.
```

if 조건문을 사용하여 같은 프로그램을 작성한다면, 더 복잡해질 것이고, 복사한 코드를 포함하게 될 것입니다.

```
var
    Meal: Byte;
begin
    Writeln('Welcome to Pascal Restaurant. Please select your order');
    Writeln('1 – Chicken        (10$)');
    Writeln('2 – Fish           (7$)');
    Writeln('3 – Meat           (8$)');
    Writeln('4 – Salad          (2$)');
    Writeln('5 - Orange Juice (1$)');
    Writeln('6 – Milk           (1$)');
    Writeln;
    Write('Please enter your selection: ');
    Readln(Meal);

    if Meal = 1 then
        Writeln('You have ordered Chicken, this will take 15 minutes')
    else
    if Meal = 2 then
        Writeln('You have ordered Fish, this will take 12 minutes')
    else
    if Meal = 3 then
        Writeln('You have ordered meat, this will take 18 minutes')
    else
    if Meal = 4 then
        Writeln('You have ordered Salad, this will take 5 minutes')
    else
    if Meal = 5 then
        Writeln('You have ordered Orange juice,' ,
          ' this will take 2 minutes')
    else
    if Meal = 6 then
        Writeln('You have ordered Milk, this will take 1 minute')
    else
        Writeln('Wrong entry');
    end;

    Write('Press enter key to close');
    Readln;
end.
```

다음 예제에서는, 프로그램이 학생들의 성적을 계산해서 A 그리고 B, C, D, E, F 등급으로 변환해줍니다.

```
var
    Mark: Integer;
begin
    Write('Please enter student mark: ');
    Readln(Mark);
    Writeln;

    case Mark of
        0 .. 39: Writeln('Student grade is: F');
        40 .. 49: Writeln('Student grade is: E');
        50 .. 59: Writeln('Student grade is: D');
        60 .. 69: Writeln('Student grade is: C');
        70 .. 84: Writeln('Student grade is: B');
        85 .. 100: Writeln('Student grade is: A');
    else
        Writeln('Wrong mark');
    end;
    Write('Press enter key to close');
    Readln;
end.
```

위의 예제에서 (0 .. 39)와 같은 범위를 사용했으며, *Mark* 값이 이 범위에 존재하면 *True*를 되돌린다는 의미를 지닙니다.

참고

Case 구문은 *정수형*과 *문자형* 같은 서수형에만 동작하고, *문자열*과 *실수*와 같은 다른 형식에는 동작하지 않습니다.

1.11 키보드 프로그램

이 예제에서는 키보드로부터 문자를 받아서 프로그램이 키보드의 입력 받은 키에 대한 줄 번호를 알려줄 것입니다.

```
var
    Key: Char;
begin
    Write('Please enter any English letter: ');
    Readln(Key);
    Writeln;

    case Key of
        'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p':
        Writeln('This is in the second row in keyboard');
        'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l':
        Writeln('This is in the third row in keyboard');
        'z', 'x', 'c', 'v', 'b', 'n', 'm':
        Writeln('This is in the fourth row in keyboard');
    else
        Writeln('Unknown letter');
    end;

    Write('Press enter key to close');
    Readln;
end.
```

참고로, case 조건문에 값 들의 집합이라는 새로운 기술을 사용했습니다.

```
        'z', 'x', 'c', 'v', 'b', 'n', 'm':
```

이는 Key를 이 값들의 집합 (z, x, c, v, b, n, m) 중 하나로 설정했다면 case 분기 구문을 실행한다는 의미입니다.

또한 다음과 같이 범위와 단일 값을 혼용할 수 있습니다.

```
        'a' ... 'd', 'x', 'y', 'z';
```

이는 값이 a와 d 사이에 있거나 x, y, z와 같을 경우 구문을 실행한다는 의미입니다.

1.12 순환문

순환문은 코드(구문)의 일부를 정해진 횟수대로 실행하거나, 조건에 만족할 때까지 실행하기 위해 사용합니다.

1.13 for 순환문

이 예제와 같이, 카운터를 사용하여 정해진 횟수대로 순환하기 위한 구문을 실행할 수 있습니다.

```
var
    i: Integer;
    Count: Integer;
begin
    Write('How many times?');
    Readln(Count);
    for i:= 1 to Count do
        Writeln('Hello there');
    Write('Press enter key to close');
    Readln;
end.
```

for 순환문 변수에는 *Integer*, *Byte* 그리고 *Char*와 같은 서수 형식을 사용합니다. 이 변수를 *루프 변수* 또는 *루프 카운터*라고 부릅니다. *루프 카운터*의 값은 임의의 숫자로 초기화할 수 있으며, *루프 카운터*의 마지막 값도 결정할 수 있습니다. 예를 들어, 5부터 10까지 세려고 한다면, 다음처럼 할 수 있습니다.

```
for i:= 5 to 10 do
```

아래 수정된 예제와 같이, 순환문을 매번 돌 때마다 *루프 카운터*의 값을 표시할 수 있습니다.

```
var
    i: Integer;
    Count: Integer;
begin
    Write('How many times?');
    Readln(Count);
    for i:= 1 to Count do
    begin
        Writeln('Cycle number: ', i);
        Writeln('Hello there');
    end;
    Write('Press enter key to close');
    Readln;
end.
```

참고로 이 시점에서 두 개의 구문을 반복하고자 이들 구문을 하나의 구문으로 만들기 위해 *begin . . end* 키워드를 사용했습니다.

1.14 for 순환문을 사용한 곱셈표

곱셈표 프로그램의 for 순환문 버전은 더 쉽고 명확합니다.

```
Program MultTableWithForLoop;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    {you can add units after this};

var
    x, i: Integer;
begin
    Write('Please input any number:');
    Readln(x);
    for i := 1 to 12 do
        Writeln(x,' * ', i , ' = ',x * i);

    Writeln('Press enter key to close');
    Readln;
end.
```

Writeln 구문을 12번 작성하는 대신, 12번 실행하는 순환문 안에 한번 작성했습니다.

이 구문으로 ***to*** 키워드 대신 ***downto*** 키워드를 사용하여 역방향으로 반복하는 *for* 순환 구문을 만들 수 있습니다.

```
for i := 12 downto 1 do
```

1.15 팩토리얼 프로그램

수학에서 팩토리얼은 숫자를 1 감소시키면서 각각의 이전의 수를 곱한 결과입니다. 예를 들어 3! = 3 * 2 * 1 = 6 입니다.

```
var
    Fac, Num, i: Integer;
begin
    Write('Please input any number:');
    Readln(Num);
    Fac := 1;

    for i := Num downto 1 do
        Fac := Fac * i;
    Writeln('Factorial of', Num , 'is', Fac);
    Writeln('Press enter key to close');
    Readln;
end.
```

1.16 repeat until 순환문

지정한 횟수만큼 반복하는 *for* 순환문과는 다르게 *repeat* 순환문은 카운터가 없습니다. 각각의 상태가 일어나기까지(True를 되돌림) 순환하고, 그 다음에는 다음 구문으로 진행합니다.

예제:

```
var
    Num: Integer;
begin
    repeat
        Write('Please input a number: ');
        Readln(Num);
    until Num <= 0
    Writeln('Finished, Press enter key to close');

    Readln;

end.
```

앞의 예제에서, 프로그램은 순환문으로 진입하고 사용자에게 숫자를 입력하라고 요구합니다. 만약 숫자가 0보다 작거나 같다면, 순환문을 빠져나갈 것입니다. 만약 입력한 값이 0보다 크다면 순환은 계속될 것입니다.

1.17 repeat 순환문을 사용한 음식점 프로그램

```
var
    Selection: Char;
    Price: Integer;
    Total: Integer;
begin
    Total := 0;
    repeat
        Writeln('Welcome to Pascal Restaurant. Please select your order');
        Writeln('1 – Chicken (10 Geneh)');
        Writeln('2 – Fish (7 Geneh)');
        Writeln('3 – Meat (8 Geneh)');
        Writeln('4 – Salad (2 Geneh)');
        Writeln('5 - Orange Juice(1 Geneh)');
        Writeln('6 – Milk (1 Geneh)');
        Writeln('X - nothing');
        Writeln;
        Write('Please enter your selection: ');
        Readln(Selection);
```

```
        case Selection of
            '1': begin
                Writeln('You have ordered Chicken, this will take 15 minutes');
                Price:= 10;
            end;
            '2': begin
                Writeln('You have ordered Fish, this will take 12 minutes');
                Price:= 7;
            end;
            '3': begin
                Writeln('You have ordered meat, this will take 18 minutes');
                Price:= 8;
            end;
            '4': begin
                Writeln('You have ordered Salad, this will take 5 minutes');
                Price:= 2;
            end;
            '5': begin
                Writeln('You have ordered Orange juice, this will take 2 minutes');
                Price:= 1;
            end;
            '6': begin
                Writeln('You have ordered Milk, this will take 1 minute');
                Price:= 1;
            end;
            else
            begin
                Writeln('Wrong entry');
                Price:= 0;
            end;
        end;
        Total:= Total + Price;
    until (Selection = 'x') or (Selection = 'X');
    Writeln('Total price = ', Total);
    Write('Press enter key to close');
    Readln;
end.
```

앞의 예제에서 이 기법들을 사용했습니다.

1. case 분기에 *begin end*를 추가하여 여러개의 구문을 하나의 구문으로 만들었습니다.
2. 주문의 총 가격을 더하기 위해 변수 *Total* 값을 0(변수에 초기값 위치)으로 초기화했습니다. 그 다음 선택한 가격을 매 순환마다 *Total*변수에 더했습니다.

```
Total := Total + Price;
```

3. 주문을 끝내기 위해 대문자 'X'와 소문자 'x' 두가지 옵션을 넣었습니다. 두 문자는 컴퓨터 메모리에서 다르게 표현(저장)합니다.

참고:

아래 줄을

```
until (Selection = 'x' or (Selection = 'X');
```

다음처럼 줄인 코드로 바꿀 수 있습니다.

```
until UpCase(Selection) = 'X';
```

이는 *Selection* 변수의 값이 소문자라면 대문자로 바꿔줄 것이고, (x또는 X) 두 경우에 대해 *True*를 되돌릴 것입니다.

1.18 while 순환문

While 순환문은 *repeat* 순환문과 유사하지만, 다음과 같은 점이 다릅니다

1. *while* 에서는 순환문에 진입하기 전에 상태를 먼저 확인하지만 *repeat*는 순환문에 먼저 진입하고 상태를 확인합니다. 이는 *repeat*는 최소한 한 번은 구문을 실행하지만, *while* 순환문은 시작시 상태문에서 *False*를 되돌린다면 첫번째 순환의 진입을 막을 수 있다는 것을 의미합니다.

2. *while* 순환문은 순환문에서 실행할 구문들이 여러개 있을 경우 *begin end*가 필요하지만, *repeat*는 *begin end*가 필요하지 않으며, 블록(반복하는 구문들)은 *repeat* 키워드부터 시작해서 *until* 키워드에서 끝납니다.

예제:

```
var
    Num: Integer;
begin
    Write('Please input a number: ');
    Readln(Num);
    while Num > 0 do
    begin
        Write('From inside loop: input a number: ');
        Readln(Num);
    end;
    Write('Press enter key to close');
    Readln;
end.
```

1.19 while 순환문을 사용한 팩토리얼 프로그램

```
var
    Fac, Num, i: Integer;
begin
    Write('Please input any number: ');
    Readln(Num);
    Fac:= 1;
    i := num;
    while i > 1 do
    begin
        Fac:= Fac * i;
        i:= i - 1;
    end;
    Writeln('Factorial of ', Num ,' is ', Fac);
    Writeln('Press enter key to close');
    Readln;
end.
```

while 순환문은 루프 카운터가 없기 때문에 변수 i를 루프 카운터처럼 동작하게끔 사용했습니다. 루프 카운터 값은 팩토리얼 값을 얻기 위한 수로 초기화 했으며, 순환할 때마다 감소했습니다. i가 1에 도달하면 순환문의 동작을 멈출 것입니다.

1.20 문자열

string 형식은 연속된 문자를 지닐 수 있는 변수를 선언하기 위해 사용합니다. 본문, 이름, 또는 자동차 등록 표지판 번호와 같은 문자와 숫자의 조합을 저장할 때 사용할 수 있습니다.

이 예제에서는 사용자 이름을 받아들이기 위한 *string* 변수를 어떻게 사용하는지 보게 될 것입니다.

```
var
    Name: string;
begin
    Write('Please enter your name: ');
    Readln(Name);
    Writeln('Hello ',Name);

    Writeln('Press enter key to close');
    Readln;
end.
```

다음 예제에서는 사람의 정보를 저장하기 위해 문자열 형의 변수들을 사용할 것입니다.

```
var
    Name: string;
    Address: string;
    ID: string;
    DOB: string;
begin
    Write('Please enter your name: ');
    Readln(Name);
    Write('Please enter your address: ');
    Readln(Address);
    Write('Please enter your ID number: ');
    Readln(ID);
    Write('Please enter your date of birth: ');
    Readln(DOB);
    Writeln('Hello ',Name);
    Writeln;
    Writeln('Card:');
    Writeln('-----------------------------------');
    Writeln('| Name : ', Name);
    Writeln('| Address : ', Address);
    Writeln('| ID : ', ID);
    Writeln('| DOB : ', DOB);
    Writeln('-----------------------------------');
    Writeln('Press enter key to close');
    Readln;
end.
```

문자열들은 긴 문자열을 만들기 위해 결합할 수 있습니다. 예를 들어, 다음 예제와 같이 *FirstName*, *SecondName*, *FamilyName*을 *FullName*이라고 하는 다른 문자열에 합칠 수 있습니다.

```
var
    YourName: string;
    Father: string;
    GrandFather: string;
    FullName: string;
begin
    Write('Please enter your first name: ');
    Readln(YourName);
    Write('Please enter your father name: ');
    Readln(Father);
    Write('Please enter your grand father name: ');
    Readln(GrandFather);
    FullName: YourName + ' ' + Father + ' ' + GrandFather;
    Writeln('Your full name is: ', FullName);
    Writeln('Press enter key to close');
    Readln;
end.
```

참고로 이 예제에서 이름들 사이를 분리하기 위해 (*YourName* + ' ' + *Father* 와 같은)이름들 사이에 공백(' ')을 더했습니다. 공백도 역시 문자입니다.

문자열에 대해 문자열에서 하위텍스트를 찾든지, 문자열 하나를 다른 문자열 변수로 복사한다든지, 또는 다음 예제와 같이 텍스트 문자들을 대문자로 혹은 소문자로 바꾸는 등의 여러가지 처리를 할 수 있습니다.

이 줄은 *FullName*에 들어있는 문자들을 대문자 문자열 값으로 바꿉니다.

```
    FullName := UpperCase(FullName);
```

그리고 이 줄은 소문자로 바꿉니다.

```
    FullName := LowerCase(FullName);
```

다음 예제에서는 *Pos* 함수를 사용하여 사용자 이름에서 *a* 문자를 검색하려고 합니다. *Pos* 함수는 문자열의 첫번째 존재(인덱스)를 되돌리거나, 검색한 텍스트에서 문자나 하위 문자열이 없는 경우 0을 반환합니다.

```
var
    YourName: string;
begin
    Write('Please enter your name: ');
    Readln(YourName);
    if Pos('a', YourName) > 0 then
        Writeln('Your name contains a');
    else
        Writeln('Your name does not contain a letter');

    Writeln('Press enter key to close');
    Readln;
end.
```

이름에 대문자 *A*를 포함하고 있다면, *Pos* 함수는 이것을 가리키지 않을 것입니다. 왜냐면 앞서 우리가 보았듯이 *A*는 *a*와 다르기 때문입니다.

이 문제를 해결하기 위해 모든 사용자 이름의 글자를 소문자로 바꾼 다음에 검색을 시작할 수 있습니다.

```
If Pos('a', LowerCase(YourName)) > 0 then
```

다음 수정한 코드에서는, 사용자 이름에 *a*의 위치가 어디에 있는지를 표시할 것입니다.

```
var
    YourName: string;
begin
    Write('Please enter your name: ');
    Readln(YourName);
    if Pos('a', YourName) > 0 then
    begin
        Writeln('Your name contains a');
        Writeln('a position in your name is: ',
          Pos('a',LowerCase(YourName)));
    end
    else
        Writeln('Your name does not contain a letter');

    Writeln('Press enter key to close');
    Readln;
end.
```

이름에 하나 이상의 *a* 글자가 들어있다면, *Pos* 함수는 사용자 이름에서 문자 *a*의 처음 위치 인덱스를 되돌릴 것임을 참고하시는 것이 좋겠습니다.

Length 함수를 사용하여 문자열에 있는 문자의 수를 셀 수 있습니다.

```
Writeln('Your name length is ',Length(YourName),' letters');
```

또한 인덱스 번호를 사용하여 문자열의 첫번째 글자/문자를 가져올 수 있습니다.

```
Writeln('Your first letter is ',YourName[1]);
```

그리고 두번째 문자도 가져올 수 있습니다.

```
Writeln('Your second letter is ',YourName[2]);
```

마지막 문자도 가져올 수 있습니다.

```
Writeln('Your last letter is ',YourName[Length(YourName)]);
```

또한 *for* 순환문을 사용하여 문자열 변수의 문자를 문자별로 보여줄 수 있습니다.

```
for i := 1 to Length(YourName) do
    Writeln(YourName[i]);
```

1.21 Copy 함수

copy 함수를 사용하여 문자열의 일부를 복사할 수 있습니다. 예를 들어 문자열 *'hello world'*에서 *'world'* 단어를 가져오려고 한다면, 아래 우리가 작성한 예제와 같이 해당 부분의 위치를 알고 있을 때 복사할 수 있습니다.

```
var
    Line: string;
    Part: string;
begin
    Line:= 'Hello world';

    Part:= Copy(Line, 7, 5);

    Writeln(Part);

    Writeln('Press enter key to close');
    Readln;
end.
```

이 구문을 통해 *Copy* 함수를 사용했습니다.

```
    Part := Copy(Line, 7, 5);
```

위 구문에 대한 설명입니다.

Part := 함수의 결과(하위 문자열 *'world'*)를 저장할 문자열 변수입니다.

Line *'Hello world'*문장이 들어있는 원본 문자열입니다.

7 뽑고 싶은 하위 문자열의 시작 지점 혹은 인덱스 입니다. 이 경우 이 위치에 있는 문자는 *'w'*입니다.

5 뽑을 부분의 길이입니다. 이 경우 단어 *'world'*의 길이를 나타냅니다.

다음 예제에서, 사용자에게 *February*와 같은 달 이름의 입력을 요구한 다음, 프로그램에서 입력한 달 이름을 *Feb*와 같은 짧은 버전으로 찍어낼 것입니다.

```
var
    Month: string;
    ShortName: string;
begin
    Write('Please input full month name e.g. January : ');
    Readln(Month);
    ShortName:= Copy(Month, 1, 3);
    Writeln(Month, ' is abbreviated as : ', ShortName);
    Writeln('Press enter key to close');
    Readln;
end.
```

1.22 Insert 프로시저

Insert 프로시저는 문자열에 하위 문자열을 넣습니다. 두 개의 하위 문자열을 연결하는 문자열 결합 연산자(+)와는 달리, *Insert*는 다른 문자열의 중간에 하위 문자열을 넣습니다.

예를 들어, 다음 예제처럼 *'Hello Pascal world'* 라는 결과가 되게 *'Hello world'* 문자열에 *'Pascal'*문자열을 넣을 수 있습니다.

```
var
    Line: string;
begin
    Line:= 'Hello world';

    Insert('Pascal ', Line, 7);

    Writeln(Line);

    Writeln('Press enter key to close');
    Readln;
end.
```

Insert 프로시저의 인자들은 다음과 같습니다

'Pascal' 대상 문자열에 넣으려는 하위 문자열입니다.

Line 처리 결과를 가지게 될 대상 문자열입니다.

7 대상 문자열에 넣을 시작 위치입니다. 이 경우 *'Hello world'*의 첫번째 공백 다음의 일곱번째 문자 자리가 되겠습니다.

1.23 Delete 프로시저

이 프로시저는 문자열로부터 문자나 하위 문자열을 삭제할 때 사용합니다. 지울 g 하위 문자열의 시작위치와 길이를 알 필요가 있습니다.

예를 들어, *'Heo World'*를 만들기 위해 *'Hello World'*로부터 *ll* 문자들을 삭제하려 한다면, 다음과 같이 할 수 있습니다.

```
var
    Line: string;
begin
    Line:= 'Hello world';
    Delete(Line, 3, 2);
    Writeln(Line);
    Writeln('Press enter key to close');
    Readln;
end.
```

1.24 Trim 함수

이 함수는 문자열의 시작부분과 끝 부분의 공백 문자를 제거하기 위해 사용합니다. 만약 ' *Hello* '가 들어있는 문자열을 가지고 있다면, 이 함수를 사용했을 때 '*Hello*'가 됩니다. 공백 사이사이로 문자들을 입력하기 전에는 터미널 창에서 공백 문자를 보여줄 수 없습니다.

```
Program TrimStr;

{$mode objfpc}{$H+};

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    Line: string;
begin
    Line:= ' Hello ';
    Writeln('<', Line, '>');
    Line:= Trim(Line);
    Writeln('<', Line, '>');
    Writeln('Press enter key to close');
    Readln;
end.
```

앞의 예제에서 *Trim* 함수가 들어있는 *SysUtils* Unit을 사용했습니다.

문자열의 앞 뒤 방향 중 한 쪽에 대해서만 공백을 제거할 수 있는 다른 두 가지 함수들이 있습니다. 그것들은 바로 *TrimRight*와 *TrimLeft*입니다.

1.25 StringReplace 함수

StringReplace 함수는 주어진 문자열에 대해 문자들이나 하위 문자열을 다른 문자나 문자열로 대체합니다.

```
Program StrReplace;

{$mode objfpc}{$H+};

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    Line: string;
    Line2: string;
begin
    Line:= 'This is a test for string replacement';
    Line2:= StringReplace(Line, ' ', '-', [rfReplaceAll]);      Writeln(Line);
    Writeln(Line2);
    Write('Press enter key to close');
    Readln;
end.
```

StringReplace 함수의 인자는 다음과 같습니다

Line 수정하려고 하는 원본 문자열입니다.

' ' 바꾸려는 하위 문자열입니다. 이 예제에서는 공백문자입니다.

'-' 원본 문자열에서 앞의 것을 바꾸려는 하위 문자열입니다.

[rfReplaceAll] 바꿀 형식입니다. 이 경우 *공백문자* 하위 문자열의 모든 위치에 대해 바꾸려고 합니다.

다음 수정한 예제에서 보듯이, 단지 하나의 문자열 변수만 사용하고 *Line2* 변수를 무시할 수도 있습니다만, 원본 텍스트 값을 잃게 될 것입니다.

```
var
    Line: string;
begin
    Line := 'This is a test for string replacement';
    Writeln(Line);
    Line := StringReplace(Line, ' ', '-', [rfReplaceAll]);
    Writeln(Line);
    Write('Press enter key to close');
    Readln;
end.
```

1.26 배열

배열은 같은 형을 지닌 변수들의 연속입니다. 10개의 Integer 변수에 대한 배열을 선언하려 한다면, 다음과 같이 할 수 있습니다.

```
    Numbers: array [1 .. 10] of Integer;
```

배열의 인덱스를 사용하여 배열의 단일 변수에 접근할 수 있습니다.

```
    Numbers[1] := 30;
```

두번째 변수에 값을 넣으려면, 인덱스 2를 사용합니다.

```
    Numbers[2] := 315;
```

다음 예제에서는, 10명의 학생들의 성적을 입력하도록 요구하고 이들을 배열에 넣습니다. 게다가 통과/탈락 결과까지도 바로 뽑아낼 것입니다.

```
var     Marks: array [1 .. 10] of Integer;
    i: Integer;
begin
    for i:= 1 to 10 do
    begin
        Write('Input student number ', i, ' mark: ');
        Readln(Marks[i]);
    end;

    for i:= 1 to 10 do
    begin
        Write('Student number ', i, ' mark is : ', Marks[i]);
        if Marks[i] >= 40 then
            Writeln(' Pass')
        else
            Writeln(' Fail');
    end;

    Writeln('Press enter key to close');
    Readln;
end.
```

최대 최소 학생 성적을 가져오기 위해 이전 코드를 수정할 수 있습니다.

```
var
    Marks: array [1 .. 10] of Integer;
    i: Integer;
    Max, Min: Integer;
begin
    for i:= 1 to 10 do
    begin
        Write('Input student number ', i, ' mark: ');
        Readln(Marks[i]);
    end;

    Max := Marks[1];
    Min := Marks[1];

    for i:= 1 to 10 do
    begin
        // Check if current Mark is maximum mark or not
        if Marks[i] > Max then
            Max := Marks[i];
        // Check if current Mark is minimum mark or not
        if Marks[i] < Min then
            Min := Marks[i];

        Write('Student number ', i, ' mark is : ', Marks[i]);

        if Marks[i] >= 40 then
            Writeln(' Pass')
        else
            Writeln(' Fail');
    end;
    Writeln('Max mark is ', Max);
    Writeln('Min mark is ', Min);
    Writeln('Press enter key to close');
    Readln;
end.
```

참고로, 첫번째 성적(*Marks[1]*)을 최대 최소 성적으로 가정하고, 나머지 성적들과 비교할 것입니다.

```
    Max := Marks[1];
    Min := Marks[1];
```

순환문 안에서 각각의 성적에 대해 *Max*와 *Min*을 비교합니다. *Max*보다 큰 수를 찾으면 *Max*의 값을 더 큰 수로 바꾸고, *Min*보다 작은 수를 찾으면 *Min*의 값을 더 작은 수로 바꿉니다.

앞의 예제에서 주석을 도입했습니다.

```
        // Check if current Mark is maximum mark or not
```

이 줄은 주석이라는 것을 의미하는 // 문자들로 시작했으며, 코드에 영향을 주지 않고 컴파일되지 않을 것입니다. 이 기술은 코드의 일부분을 다른 프로그래머들에게 설명하거나 프로그래머 자신에게 설명할 때 사용합니다.

// 는 짧은 주석에 사용합니다. 주석을 여러 줄로 작성하려 한다면, *{ }* 또는 *(**)* 로 주석으로 삼을 문자열들을 감쌀 수 있습니다.

```
    for i:= 1 to 10 do

    begin
        { Check if current Mark is maximum mark or not
        check if Mark is greater than Max then put
        it in Max }
        if Marks[i] > Max then
            Max := Marks[i];
        (* Check if current value is minimum mark or not
        if Min is less than Mark then put Mark value in Min
        *)
        if Marks[i] < Min then
            Min := Marks[i];
        Write('Student number ', i, ' mark is : ', Marks[i]);
        if Marks[i] >= 40 then
            Writeln(' Pass')
        else
            Writeln(' Fail');
    end;
```

또한 코드 일부를 주석처리하여 일시적으로 비활성화 할 수 있습니다.

```
    Writeln('Max mark is ', Max);
    // Writeln('Min mark is ', Min);
    Writeln('Press enter key to close');
    Readln;
```

위의 코드에서 학생 최소 성적을 기록하는 프로시저를 비활성화 했습니다.

참고

C언어에서와 같이 0을 기반으로 하는 인덱스를 가진 배열을 선언할 수 있습니다.

```
    Marks: array [0 .. 9] of Integer;
```

이 배열도 10개의 요소를 지닐 수 있지만, 첫번째 항목은 0번 인덱스를 사용하여 접근할 수 있습니다.

```
Numbers[0] := 30;
```

두번째 항목

```
Numbers[1] := 315;
```

마지막 항목

```
Numbers[9] := 10;
```

또는 다음과 같이 접근하기도 합니다.

```
Numbers[High(Numbers)] := 10;
```

1.27 레코드

*배열*은 같은 형의 많은 변수들을 지닐 수 있지만, *레코드*는 제각기 다른 형의 변수를 지닐 수 있으며, 이 변수들을 '*필드*'라고 부릅니다.

이 변수/필드의 모임은 단일 단위나 변수처럼 취급합니다. 예를 들어 자동차 정보와 같은 것들을 하나의 객체에 주려 할 때 레코드를 사용할 수 있습니다.

1. 자동차 형식 : 문자열 변수
2. 엔진 크기 : 실수
3. 생산 연도 : 정수 값

다음 예제에서는 제각각 다른 이 형식들을 자동차를 나타내는 레코드에 모을 수 있습니다.

```
Program Cars;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

type
    TCar = record
        ModelName: string;
        Engine: Single;
        ModelYear: Integer;
    end;
var
    Car: Tcar;
begin
    Write('Input car Model Name: ');
    Readln(Car.ModelName);
    Write('Input car Engine size: ');
    Readln(Car.Engine);
    Write('Input car Model year: ');
    Readln(Car.ModelYear);
    Writeln;
```

```
    Writeln('Car information: ');
    Writeln('Model Name : ', Car.ModelName);
    Writeln('Engine size : ', Car.Engine);
    Writeln('Model Year : ', Car.ModelYear);
    Write('Press enter key to close..');
    Readln;
end.
```

이 예제에서는 *'type'* 키워드를 사용하여 새 형식(*record*)을 정의했습니다.

```
type
    TCar = record
        ModelName: string;
        Engine: Single;
        ModelYear: Integer;
    end;
```

변수가 아니라 형식이라는 것을 표시하기 위해 *Car*에 문자(*T*)를 더했습니다. 변수 이름은 *Car*, *Hour*, *UserName*처럼 될 수 있지만, 형식 이름은 *TCar*, *THour*, *TUserName*과 같이 되어야 합니다. 이것은 파스칼 언어의 표준입니다.

이 새로운 형을 사용하려면, 이 형의 변수를 예제와 같이 선언할 것입니다.

```
var
    Car: Tcar;
```

이 변수/필드들 중 하나에 값을 저장하려면 다음과 같이 접근합니다.

```
    Car.ModelName
```

레코드는 이 책의 *임의 접근 파일* 편에서 사용할 것입니다.

1.28 파일

파일을 운영체제와 프로그램의 중요한 요소입니다. 운영체제 구성요소들은 파일로 나타나며 정보나 데이터 역시 사진이나 책, 프로그램, 간단한 텍스트 파일처럼 파일로 나타냅니다.

운영체제는 읽기, 쓰기, 편집, 삭제와 같은 파일 관리를 제어합니다.

파일들은 수많은 관점에 따라 다양한 형태로 나눕니다. 파일은 *실행* 파일과 *데이터* 파일 두가지 형태로 나누어 볼 수 있습니다. 예를 들어 컴파일된 이진 라자루스 프로그램들은 실행 파일이고 파스칼 소스 코드(.pas)들은 데이터 파일입니다. 또한 PDF 책, JPEG 그림들도 데이터 파일입니다.

데이터 파일을 내용 표현에 따라 두가지로 나누어 볼 수 있습니다.

1. *텍스트 파일* : 기록할 수 있거나 리눅스의 *cat, vi* 그리고 윈도우즈의 *type, copy con*과 같은 명령으로 운영체제 명령줄을 포함한 임의의 간단한 도구를 사용하여 읽어들일 수 있는 단순한 텍스트 파일입니다.
2. *바이너리 데이터 파일* : 훨씬 복잡하며 이들을 열기 위해 특별한 프로그램이 필요합니다. 예를 들어 그림 파일은 단순한 명령줄 도구로 열 수 없으며, 대신 *GIMP, Kolour Paint, MSPaint* 등과 같은 프로그램을 사용하여 엽니다. 그림이나 음성 파일을 연다면, 사용자들이 인지할 수 없는 의미없는 문자들을 보게 될 것입니다. 이진 데이터 파일의 예로 적당한 프로그램을 사용하여 열어야 하는 데이터베이스 파일이 있습니다.

접근 형태에 따라 다른 방법으로 파일을 분류할 수 있습니다.

1. *순차 접근 파일* : 순차 접근 파일의 예로 고정되지 않은 크기의 레코드를 지닌 텍스트 파일이 있습니다. 제각각의 줄은 고유의 길이를 가지고 있어, 예로 들자면, (문자 단위의)세번째 줄의 시작 위치를 알 수 없습니다. 이러한 이유로 읽기 전용 또는 쓰기 전용으로 파일을 열 수 있으며, 파일의 마지막에 텍스트만 붙일 수 있습니다. 만약 파일의 중간에 텍스트를 넣고 싶다면, 파일의 내용을 메모리로 읽어야 하고, 수정을 가하고, 전체 파일을 디스크에서 지운 다음에, 수정된 텍스트를 파일에 덮어써야 합니다.
2. *임의 접근 파일* : 이 파일의 형식은 고정된 크기의 레코드를 가집니다. 레코드는 우리가 한번에 읽고 쓸 수 있는 작은 단위며, 바이트, 정수형, 문자열 또는 사용자가 지정한 레코드일 수도 있습니다. 동시에 읽고 쓸 수 있는데, 예를 들자면, 3번 레코드에서 읽어들여서 10번 레코드에 복사할 수 있습니다. 이 경우, 파일의 각각의 레코드에 대한 위치를 정확하게 알 수 있습니다. 레코드를 수정하는 것은 간단합니다. 임의 접근 파일에서는 파일의 나머지 부분에 영향을 주지 않고 그 어떤 레코드든지 바꾸고 덮어쓸 수 있습니다.

1.29 텍스트 파일

텍스트 파일은 가장 간단한 파일이지만, 한 방향(진행 방향)으로 기록할 것입니다. 텍스트 파일을 기록하는 동안 뒤로 돌아갈 수 없습니다. 또한 파일을 열 때 읽기, 쓰기, 추가(뒷 부분에 기록)중 하나의 처리 모드로 정의해주어야 합니다.

이 예제에서는 사용자가 선택한 텍스트 파일의 내용을 표시할 것입니다. 예를 들어, 사용자가 윈도우즈에서 C:\test\first.pas, 또는 리눅스에서 /home/user/first.pas 와 같은 파일을 가지고 있다고 가정해봅니다.

1.30 텍스트 파일 읽기 프로그램

```
Program ReadFile;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    FileName: string;
    F: TextFile;
    Line: string;
begin
    Write('Input a text file name: ');
    Readln(FileName);
    if FileExists(FileName) then
    begin
        // Link file variable (F) with physical file (FileName)
        AssignFile(F, FileName);
        Reset(F); // Put file mode for read, file should exist

        // while file has more lines that does not read yet do the loop
        while not Eof(F) do
        begin
            Readln(F, Line); // Read a line from text file
            Writeln(Line); // Display this line in user screen
        end;
        CloseFile(F); // Release F and FileName connection
    end
```

```
    else // else if FileExists..
        Writeln('File does not exist');
    Write('Press enter key to close..');
    Readln;
end.
```

리눅스에서는 다음 파일 이름

```
/etc/resolv.conf
```

또는 다음 파일 이름을 입력할 수 있습니다.

```
/proc/meminfo
/proc/cpuinfo
```

텍스트 파일을 다루기 위해 새로운 형과 함수, 프로시저를 사용했는데,

1.
```
    F: TextFile
```

*TextFile*은 텍스트 파일 변수를 선언하기 위해 사용하는 형입니다. 이 변수는 다음의 사용을 위해 텍스트 파일에 연결할 수 있습니다.

2.
```
    if FileExists(FileName) then
```

*FileExists*는 *SysUtils* Unit에 들어있는 함수입니다. 파일의 존재를 확인합니다. 저장 매체에 파일이 존재하는 경우 *True*를 되돌립니다.

3.
```
    AssignFile(F, FileName);
```

파일의 존재를 확인한 다음에는, 파일 변수(F)에 실제 파일을 연결하기 위해 *AssignFile* 프로시저를 사용할 수 있습니다. F 변수의 어떤 그 이상의 사용방법은 실제 파일에 따라 달려있습니다.

4.
```
Reset(F); // Put file mode for read, file should exist
```

Reset 프로시저는 텍스트 파일을 읽기 전용으로 읽어들이고 파일의 첫번째 문자에 읽기 포인터를 위치합니다.
만약 현재 사용자에게 파일의 읽기 권한이 없다면, *접근 거부*(*Access denied*)와 같은 오류 메시지가 나타날 것입니다.

5.
```
Readln(F, Line); // Read a line from text file
```

Readln 프로시저는 파일로부터 완전한 한 줄을 읽어들여 변수 *Line*에 넣기위해 사용합니다. 텍스트 파일로부터 한 줄을 읽기 전에 파일의 끝에 도달했는지 확인할 것입니다. 우리는 다음 함수 *Eof*를 통해 이를 해결할 수 있습니다.

6.
```
while not Eof(F) do
```

Eof 함수는 파일의 끝에 도달했을 경우 *True*를 되돌립니다. Read 처리를 더 이상 사용할 수 없고 파일의 내용을 다 읽었다는 상태를 나타낼 때 사용합니다.

7.
```
CloseFile(F); // Release F and FileName connection
```

파일로부터 읽어오는 동작이나 파일에 기록하는 동작이 끝나면, 파일을 릴리스하기 위해 닫아야 하는데, *Reset* 프로시저가 운영체제로부터 파일을 예약하고 있고, 파일이 열려있는 동안 다른 프로그램이 기록하거나 지우는 것을 막고 있기 때문입니다.

파일을 여는 중에 *Reset* 프로시저의 동작을 성공적으로 수행했을 때, *CloseFile*을 사용할 것입니다. 만약 *Reset*이 실패(예를 들어, 파일이 존재하지 않거나 다른 프로그램이 파일을 사용하고 있다면)하면 이런 경우 파일을 닫지 않습니다.

다음 예제에서는 새로운 텍스트 파일을 만들고 임의의 텍스트를 넣어보겠습니다.

1.31 텍스트 파일 만들고 기록하기

```
var
    FileName: string;
    F: TextFile;
    Line: string;
    ReadyToCreate: Boolean;
    Ans: Char;
    i: Integer;
begin
    Write('Input a new file name: ');
    Readln(FileName);
    // Check if file exists, warn user if it is already exist
    if FileExists(FileName) then
    begin
        Write('File already exist, did you want to overwrite it? (y/n)');
        Readln(Ans);
        if upcase(Ans) = 'Y' then
            ReadyToCreate:= True
        else
            ReadyToCreate:= False;
    end
    else // File does not exist
        ReadyToCreate:= True;

    if ReadyToCreate then
    begin
        // Link file variable (F) with physical file (FileName)
        AssignFile(F, FileName);
        Rewrite(F); // Create new file for writing
        Writeln('Please input file contents line by line, '
            , 'when you finish write % then press enter');
        i:= 1;
        repeat
            Write('Line # ', i, ':');
            Inc(i);
            Readln(Line);
            if Line <> '%' then
                Writeln(F, Line); // Write line into text file
        until Line = '%';
        CloseFile(F); // Release F and FileName connection, flush buffer
    end
    else // file already exist and user does not want to overwrite it
        Writeln('Doing nothing');
    Write('Press enter key to close..');
    Readln;
end.
```

이 예제에서는 많은 중요한 요소들을 사용했습니다.

1. *Boolean* 형:

```
ReadyToCreate: Boolean;
```

이 형은 *True*값과 *False*값 둘 중 하나만 지닐 수 있습니다. 이 값들은 *if* 조건문, *while* 순환문 또는 *repeat* 순환문에서 바로 사용할 수 있습니다.

앞의 예제에서 다음과 같은 if 조건문을 사용했습니다.

```
if Marks[i] > Max then
```

여기서도 *True* 또는 *False*를 되돌립니다.

2. *UpCase* 함수:

```
if upcase(Ans) = 'Y' then
```

이 구문은 파일이 존재할 때 실행합니다. 프로그램은 기존의 파일에 덮어쓸 것인지에 대한 여부를 사용자에게 경고할 것입니다. 사용자가 계속하려고 한다면, 소문자 *y*나 대문자 *Y*를 입력할 것입니다. *UpCase* 함수는 입력한 문자가 소문자인 경우 대문자로 바꿀 것입니다.

3. *Rewrite* 프로시저:

```
Rewrite(F); // Create new file for writing
```

Rewrite 프로시저는 새로운 빈 파일을 만들 때 사용합니다. 파일이 이미 존재한다면, 지우고 덮어쓸 것입니다. 텍스트 파일인 경우 쓰기 전용으로도 엽니다.

4. *Writeln(F, ..)* 프로시저:

```
Writeln(F, Line); // Write line into text file
```

이 프로시저는 텍스트 파일에 문자열이나 변수를 기록하고, 이들 데이터에 숫자 13과 10으로 각각 표현되는 캐리지 리턴/라인 피드의 조합인 줄의 마지막 문자를 붙이기 위해 사용합니다.

이 문자들은 콘솔 창에 표시할 수 없지만, 화면에 나타나는 커서를 새로운 줄로 이동할 것입니다.

5. Inc 프로시저:

```
Inc(i);
```

이 프로시저는 정수형 변수를 1만큼 증가합니다. 다음 구문과 같습니다.

```
i := i + 1;
```

6. CloseFile 프로시저:

```
CloseFile(F); // Release F and FileName connection, flush buffer
```

앞서 보신 바와 같이, *CloseFile* 프로시저는 운영체제로 파일을 돌려줍니다. 덧붙여서, 텍스트 파일에 기록할 때 기록버퍼를 비우는 추가적인 작업을 수행합니다.

텍스트 파일의 버퍼링은 텍스트 파일을 빠르게 다루게 해주는 기능입니다. 단일 줄이나 문자를 디스크나 다른 저장 매체에 바로 기록(*메모리에 기록하는 것과 비교했을 때 매우 느립니다*)하는 대신에, 프로그램은 이 항목들을 메모리 버퍼에 기록할 것입니다. 버퍼가 거의 가득 찼을 때, 하드 디스크와 같은 영구 저장 매체에 몰아넣습니다(강제로 기록합니다). 이 동작은 기록을 빠르게 해주지만, 전원이 갑자기 나갔을 경우 (버퍼에 있는)데이터들을 잃을 수 있는 위험성을 더할 것입니다. 데이터 손실을 최소화하기 위해, 파일에 기록을 끝내고 나서 바로 파일을 닫아주거나 버퍼를 명시적으로 비우기 위해 *Flush* 프로시저를 호출해야 합니다.

1.32 텍스트 파일에 덧붙이기

이 예제에서는, *Append* 프로시저를 사용하여 기존의 내용을 지우지 않고, 기존의 텍스트 파일의 마지막 부분에 기록하기 위해 열려고 합니다.

1.33 텍스트 파일에 추가하기 프로그램

```
var
    FileName: string;
    F: TextFile;
    Line: string;
    i: Integer;
begin
    Write('Input an existed file name: ');
    Readln(FileName);
    if FileExists(FileName) then
    begin
        // Link file variable (F) with physical file (FileName)
        AssignFile(F, FileName);

        Append(F); // Open file for appending

        Writeln('Please input file contents line by line',
            'when you finish write % then press enter');
        i:= 1;
        repeat
            Write('Line # ', i, ' append :');
            Inc(i);
            Readln(Line);
            if Line <> '%' then
                Writeln(F, Line); // Write line into text file
        until Line = '%';
        CloseFile(F); // Release F and FileName connection, flush buffer
    end
    else
        Writeln('File does not exist');
    Write('Press enter key to close..');
    Readln;
end.
```

이 프로그램을 실행하고 기존의 텍스트 파일의 이름을 입력하고 나면, 이에 대한 결과는 *cat / type* 명령으로, 또는 붙은 데이터를 보기 위해 디렉터리에서 이 파일을 마우스로 두 번 눌러서 볼 수 있습니다.

1.34 임의 접근 파일

앞에서 보았듯이, 접근 형태 양상에 따른 두번째 파일 형식은 *임의 접근* 또는 *직접 접근*입니다. 이 파일의 형식은 정해진 크기의 레코드를 지니고 있어, 언제든지 읽거나 쓰기 위해 임의의 레코드로 건너뛸 수 있습니다.

임의 접근 파일의 형식은 *형식적 파일*과 *비형식적 파일* 두 가지가 있습니다.

1.35 형식적 파일

형식적 파일은 같은 크기의 레코드를 지닌 같은 형의 데이터가 들어있는 파일에 대해 사용하는데, 예를 들어 파일에 *Byte* 형의 레코드가 있다면, 이는 레코드 크기가 1 바이트 임을 의미합니다. 파일에 실수(*Single*)가 들어있다면, 레코드의 크기가 4 바이트 임을 의미하며, 다른 경우에도 마찬가지입니다.

다음 예제를 통해 file of Byte를 어떻게 사용하는지 보여드리겠습니다.

1.36 성적 프로그램

```
var
    F: file of Byte;
    Mark: Byte;
begin
    AssignFile(F, 'marks.dat');
    Rewrite(F); // Create file
    Writeln('Please input students marks, write 0 to exit');
    repeat
        Write('Input a mark: ');
        Readln(Mark);
        if Mark <> 0 then // Don't write 0 value
            Write(F, Mark);
    until Mark = 0;
    CloseFile(F);
    Write('Press enter key to close..');
    Readln;
end.
```

이 예제에서 형식적 파일을 정의하기 위해 이런 구문을 사용했습니다.

```
    F: file of Byte;
```

이는 파일에 0부터 255까지의 값을 지닐 수 있는 Byte 데이터 형을 지닌 레코드를 포함하고 있다는 뜻입니다.

그리고 *Rewrite* 프로시저를 사용하여 파일을 만들고 기록하기 위해 엽니다.

```
F: file of Byte;
```

또한 형식적 파일에 레코드를 기록하기 위해 *Writeln* 대신 *Write* 함수를 사용했습니다.

```
Write(F, Mark);
```

*Writeln*은 형식적 파일에 적당하지 않은데, *Writeln*은 기록한 텍스트의 마지막에 CR/LF 문자를 덧붙이지만, *Write*는 어떤 덧붙임 없이 레코드에 저장하기 때문입니다. 이 경우 10 레코드(바이트 단위)를 입력했다면, 파일 크기는 디스크에서 10 바이트를 차지할 것입니다.

다음 예제를 통해 파일의 내용을 어떻게 표시할 지 보여드리겠습니다.

1.37 학생 성적 읽기

```
Program ReadMarks;

{$mode objfpc}{$H}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    F: file of Byte;
    Mark: Byte;
begin
    AssignFile(F, 'marks.dat');
    if FileExists('marks.dat') then
    begin
        Reset(F); // Open file
        while not Eof(F) do
        begin
            Read(F, Mark);
            Writeln('Mark: ', Mark);
        end;
        CloseFile(F);
    end
```

```
    else
        Writeln('File (marks.dat) not found');

    Write('Press enter key to close..');
    Readln;
end.
```

다음 예제에서는 기존의 데이터를 삭제하지 않고 새 레코드를 덧붙이는 방법을 보여드리겠습니다.

1.38 학생 성적 추가 프로그램

```
Program AppendMarks;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    F: file of Byte;
    Mark: Byte;
begin
    AssignFile(F, 'marks.dat');
    if FileExists('marks.dat') then
    begin
        FileMode:= 2;// Open file for read/write
        Reset(F); // open file
        Seek(F, FileSize(F)); // Go to beyond last record
        Writeln('Please input students marks, write 0 to exit');

        repeat
            Write('Input a mark: ');
            Readln(Mark);
            if Mark <> 0 then // Don't write 0 value in disk
                Write(F, Mark);
        until Mark = 0;
        CloseFile(F);
    end
    else
        Writeln('File marks.dat not found');

    Write('Press enter key to close..');
    Readln;
end.
```

이 프로그램을 실행하고 새 레코드를 입력하고 나서 덧붙인 데이터를 보기 위해 다시 실행할 수 있습니다.

참고로 파일에 기록을 목적으로 파일을 열기 위해 *Rewrite* 프로시저 대신 *Reset* 프로시저를 사용하였습니다. *Rewrite*는 기존의 파일의 모든 데이터를 지우고, 파일이 존재하지 않는다면 빈 파일을 만들지만, *Reset*은 내용을 지우지 않고 기존의 파일을 열기만 할 수 있습니다.

또한 읽기/쓰기 접근 모드로 파일을 열려고 한다는 것을 나타내기 위해 *FileMode* 변수에 2를 할당했습니다. *FileMode*의 0은 읽기 전용, 1은 쓰기 전용, 2(기본)는 읽기/쓰기를 의미합니다.

```
FileMode:= 2;// Open file for read/write
Reset(F); // open file
```

*Reset*은 첫번째 레코드에 읽기/쓰기 포인터를 놓고, 이러한 이유로 파일에 바로 기록을 시작하면 이전 레코드에 덮어쓰게 되기 때문에, 파일의 끝으로 읽기/쓰기 포인터를 이동하기 위해 *Seek* 프로시저를 사용했습니다. *Seek*는 임의 접근 파일에서만 사용할 수 있습니다.

Seek 프로시저로 존재하지 않는 레코드 위치에 접근하려 한다면 (예를 들어, 50 레코드만 있는데 100번째 레코드에 접근하려 한다면), 오류를 만나게 될 것입니다.

파일의 현재 레코드 갯수를 되돌리는 *FileSize* 함수도 사용했습니다. 파일의 끝으로 건너뛰기 위해 *Seek* 프로시저와 함께 사용했습니다.

```
Seek(F, FileSize(F)); // Go to beyond last record
```

참고로 이 예제는 기존의 학생 성적 파일이 있을 경우에 사용할 수 있습니다. 만약 그렇지 않다면, 새 파일을 만들 수 있도록 *Rewrite*를 사용하기 때문에 처음 프로그램 (학생 성적 저장하기)을 실행해야 합니다.

다음 예제를 통해 우리가 해낸 바와 같이, 파일의 존재 유무에 따라 두 가지 방식 (*Reset*과 *Rewrite*)을 혼용할 수 있습니다.

1.39 학생 성적을 새로 만들고 덧붙이는 프로그램

```
Program ReadWriteMarks;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    F: file of Byte;
    Mark: Byte;
```

```
begin
    AssignFile(F, 'marks.dat');
    if FileExists('marks.dat') then
    begin
        FileMode:= 2; // Open file for read/write
        Reset(F); // open file
        Writeln('File already exist, opened for append');
        // Display file records
        while not Eof(F) do
        begin
            Read(F, Mark);
            Writeln('Mark: ', Mark);
        end
    end
    else // File not found, create it
    begin
        Rewrite(F);
        Writeln('File does not exist, not it is created');
    end;

    Writeln('Please input students marks, write 0 to exit');
    Writeln('File pointer position at record # ', FilePos(f));
    repeat
        Write('Input a mark: ');
        Readln(Mark);
        if Mark <> 0 then // Don't write 0 value
            Write(F, Mark);
    until Mark = 0;
    CloseFile(F);

    Write('Press enter key to close..');
    Readln;
end.
```

참고로 이 예제에서는 *Seek* 프로시저를 사용하지 않고, 대신 모든 파일의 내용을 먼저 읽었습니다. 이 동작(모든 파일을 읽기)은 포인터를 파일의 마지막으로 이동합니다.

다음 예제에서는 자동차의 정보를 저장하기 위해 레코드로 이루어진 파일을 사용할 것입니다.

1.40 자동차 데이터베이스 프로그램

```
Program CarRecords;

{$mode objfpc}{$H+}
```

```
uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

type
    TCar = record
        ModelName: string[20];
        Engine: Single;
        ModelYear: Integer;
    end;
var
    F: file of TCar;
    Car: TCar;
begin
    AssignFile(F, 'cars.dat');

    if FileExists('cars.dat') then
    begin
        FileMode:= 2; // Open file for read/write

        Reset(F); // open file
        Writeln('File already exist, opened for append');

        // Display file records
        while not Eof(F) do
        begin
            Read(F, Car);
            Writeln;
            Writeln('Car # ', FilePos(F), ' ----------');
            Writeln('Model : ', Car.ModelName);
            Writeln('Year : ', Car.ModelYear);
            Writeln('Engine: ', Car.Engine);
        end
    end
    else // File not found, create it
    begin
        Rewrite(F);
        Writeln('File does not exist, created');
    end;

    Writeln('Please input car informaion, ',
        'write x in model name to exit');
    Writeln('File pointer position at record # ', FilePos(f));
```

```
    repeat
        Writeln('————————————————');
        Write('Input car Model Name : ');
        Readln(car.ModelName);
        if Car.ModelName <> 'x' then
        begin
            Write('Input car Model Year : ');
            Readln(car.ModelYear);
            Write('Input car Engine size: ');
            Readln(car.Engine);
            Write(F, Car);
        end;
    until Car.ModelName = 'x';
    CloseFile(F);

    Write('Press enter key to close..');
    Readln;
end.
```

앞의 예제에서는 자동차 정보를 정의하기 위해 *TCar* 형을 선언했습니다. 첫번째 필드(*ModelName*)는 문자열 변수지만, 최대 길이[20]를 제한했습니다.

```
    ModelName: string[20];
```

파일에서 사용하기 전에 선언한 문자열 변수를 사용할텐데, 기본 *ANSI* 문자열 변수는 메모리에서 제각각의 제한이 없는 저장 형태를 지니고 있지만, 형식적 파일에 대해서는 모든 데이터 형의 폭을 정하는 것이 좋기 때문입니다.

1.41 파일 복사

텍스트 파일들, 이진 파일들과 같은 모든 파일의 형식은 컴퓨터 메모리와 디스크 상에서 데이터를 표현하는 가장 작은 단위인 바이트 단위를 기반으로 합니다. 모든 파일들은 1바이트, 2바이트 등을 포함하거나 또는 아무 것도 포함하지 않을 것입니다. 모든 바이트에는 0부터 255까지의 정수 값이나 문자 코드 값을 지닐 수 있습니다. *file of Byte*나 *file of Char* 선언을 사용하여 모든 파일 형식을 열 수 있습니다.

file of Byte 파일을 사용하여 어떤 파일을 다른 파일로 복사할 수 있고, 그 결과는 원본 파일의 내용과 동일한 새로운 파일이 될 것입니다.

1.42 file of Byte를 사용하여 파일 복사하기

```
Program FilesCopy;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    SourceName, DestName: string;
    SourceF, DestF: file of Byte;
    Block: Byte;
begin
    Writeln('Files copy');
    Write('Input source file name: ');
    Readln(SourceName);
    Write('Input destination file name: ');
    Readln(DestName);
    if FileExists(SourceName) then
    begin
        AssignFile(SourceF, SourceName);
        AssignFile(DestF, DestName);

        FileMode:= 0;          // open for read only

        Reset(SourceF); // open source file
        Rewrite(DestF); // Create destination file

        // Start copy
        Writeln('Copying..');
```

```
        while not Eof(SourceF) do
        begin
            Read(SourceF, Block); // Read Byte from source file
            Write(DestF, Block); // Write this byte into new
                                 // destination file
        end;
        CloseFile(SourceF);
        CloseFile(DestF);
    end

    else // Source File not found
        Writeln('Source File does not exist');
    Write('Copy file is finished, press enter key to close..');
    Readln;
end.
```

앞의 예제를 실행하면, 기존의 파일 이름과 새 대상 파일 이름을 입력할 것입니다. 리눅스에서는 파일 이름을 다음처럼 입력할 수 있습니다.

```
Input source file name: /home/motaz/quran/mishari/32.mp3
Input destination file name: /home/motaz/Alsajda.mp3
```

윈도우즈에서는 다음처럼 입력할 수 있습니다.

```
Input source file name: c:\photos\mypphoto.jpg
Input destination file name: c:\temp\copy.jpg
```

만약 *FileCopy* 프로그램과 같은 디렉터리에 원본 파일이 있다면, 다음처럼 파일 이름만 입력할 수 있습니다.

```
Input source file name: test.pas Input destination file name: testcopy.pas
```

큰 파일을 복사하기 위해 이 방법을 사용하면, 운영체제의 복사 프로시저와 비교해서 매우 많은 시간이 걸릴 것입니다. 이는 운영체제가 파일을 복사하기 위해 다른 기술을 사용한다는 것을 의미합니다. 만약 1메가바이트 파일을 복사하려 한다면, 1백만 번 *while* 순환문을 반복해야 한다는 것인데, 이는 곧 백만 번 읽고 백만 번 쓰는 동작을 수행한다는 것입니다. *file of Byte*를 *file of Word* 선언으로 대체하면, 읽기 쓰기를 500,000 번 수행하는 의미가 되지만, 파일 크기가 홀수가 아닌 짝수일 경우에만 동작할 것입니다. 만약 파일이 1,420 바이트를 가지고 있다면 잘 동작하겠지만, 1,423 바이트를 가지고 있다면 실패할 것입니다.

더 빠른 방법으로 어떤 종류의 파일이든 복사하기 위해, *비형식적* 파일을 사용할 것입니다.

1.43 비형식적 파일

비형식적 파일은 고정 레코드 길이를 지닌 임의 접근 파일이지만, 어떤 데이터 형식과도 연결된 것은 아닙니다. 대신에, 바이트나 문자의 배열처럼 데이터(*레코드*)를 다룹니다.

1.44 비형식적 파일을 사용하는 파일 복사 프로그램

```
Program FilesCopy2;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    SourceName, DestName: string;
    SourceF, DestF: file;
    Block: array [0 .. 1023] of Byte;
    NumRead: Integer;
begin
    Writeln('Files copy');
    Write('Input source file name: ');
    Readln(SourceName);
    Write('Input destination file name: ');
    Readln(DestName);

    if FileExists(SourceName) then
    begin
        AssignFile(SourceF, SourceName);
        AssignFile(DestF, DestName);

        FileMode:= 0;          // open for read only
        Reset(SourceF, 1); // open source file
        Rewrite(DestF, 1); // Create destination file
```

```
        // Start copy
        Writeln('Copying..');
        while not Eof(SourceF) do
        begin
            // Read Byte from source file
            BlockRead(SourceF, Block, SizeOf(Block), NumRead);
            // Write this byte into new destination file
            BlockWrite(DestF, Block, NumRead);
        end;
        CloseFile(SourceF);
        CloseFile(DestF);
    end
    else// Source File not found
        Writeln('Source File does not exist');

    Write('Copy file is finished, press enter key to close..');
    Readln;
end.
```

앞의 예제에서의 새로운 요소는

1. 파일의 선언 형

```
    SourceF, DescF: file;
```

2. 읽기/쓰기 변수(Buffer)

```
    Block: array [0 .. 1023] of Byte;
```

3. 추가 인자를 필요로 하는 비형식적 파일 열기

```
    Reset(SourceF, 1); // open source file
    Rewrite(DestF, 1); // Create destination file
```

추가 인자는 한 번에 읽고 쓸 수 있는 가장 작은 요소인 레코드의 크기 입니다. 임의의 파일 형식을 복사하려 할 때 1 바이트일 수도 있다는 의미인데, 임의의 파일 크기에 대해 사용할 수 있기 때문입니다.

4. Read 프로시저

```
    BlockRead(SourceF, Block, SizeOf(Block), NumRead);
```

BlockRead 프로시저는 비형식적 파일에 사용합니다. 한 번에 많은 양의 데이터를 읽습니다.

BlockRead 프로시저의 인자는 다음과 같습니다.

- SourceF : 복사하려는 원본 파일 변수입니다.
- Block : 읽고 쓸 현재 데이터를 저장할 변수나 배열입니다.
- SizeOf(Block) : 한번에 읽을 레코드의 수를 결정하는 값입니다. 예를 들어 100을 입력했다면, 100 개의 레코드(이 경우의 단위는 바이트입니다)를 읽으려 한다는 의미입니다. *SizeOf* 함수를 사용한다면, 보통 컨테이너(바이트 배열)의 수만큼 레코드를 읽으려 한다는 의미입니다.
- NumRead : 정해진 레코드 수(1024) 만큼 읽어들이라고 *BlockRead* 함수에 알렸고, 때로는 모든 레코드 수만큼 읽어들이기도 하지만, 일부만 읽어들이기도 합니다. 예를 들어 100 바이트만 들어있는 파일을 읽어들이려 한다면, *BlockRead*는 100바이트만 읽을 수 있다는 뜻입니다. 또한 정해진 양보다 적은 레코드를 읽어들이는 것은 파일의 마지막 블록에서도 일어날 수 있습니다. 예를 들어, 만약 1034 바이트가 들어있는 파일을 읽어들인다면, 처음 순환 과정에서는 1024 바이트를 읽어들이겠지만, 그 다음 순환 과정에서는 남은 10바이트만 읽어들일 것이고 *Eof* 함수는 True를 되돌릴 것입니다. *BlockWrite* 프로시저와 함께 사용하려면 *NumRead* 값이 필요합니다.

5. 비형식적 파일에 기록

```
BlockWrite(DestF, Block, NumRead);
```

쓰기 프로시저이며, *BlockRead* 프로시저와 유사하지만, 몇 가지 차이점이 있습니다.

(a) *SizeOf* 프로시저를 사용하는 대신에 *NumRead*를 사용했는데, *NumRead*에 실제 읽어들인 블록 크기가 들어있기 때문입니다

(b) 네번째 인자인 *NumWritten*(여기 예제에는 없습니다)은 중요하지 않습니다. 디스크가 가득 차기 전까지는 우리가 정한 대로 항상 기록한 레코드를 가져오기 때문입니다.

이 프로그램을 실행한 후, 큰 파일을 복사하는 속도에 주목해봅니다. 파일에 1메가바이트가 들어있다면, 전체 파일을 복사하는데 읽기/쓰기에 필요한 횟수가 대략 천 번 정도밖에 되지 않습니다.

다음 예제에서는 임의의 파일을 읽기 위해 비형식적 파일을 사용할 것이며, 메모리나 디스크에 저장한 내용을 보여줄 것입니다. 아시는 바와 같이, 파일은 바이트의 모임입니다.

1.45 파일 내용 표시 프로그램

```
Program ReadContents;

{$mode objfpc}{$H+}
```

```
uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    FileName: string;
    F: file;
    Block: array [0 .. 1023] of Byte;
    i, NumRead: Integer;
begin
    Write('Input source file name: ');
    Readln(FileName);

    if FileExists(FileName) then
    begin
        AssignFile(F, FileName);

        FileMode:= 0;          // open for read only
        Reset(F, 1);
        while not Eof(F) do
        begin
            BlockRead(F, Block, SizeOf(Block), NumRead);
            // display contents in screen
            for i:= 0 to NumRead - 1 do
                Writeln(Block[i], ':', Chr(Block[i]));
        end;
        CloseFile(F);
    end
    else// File does not exist
        Writeln('Source File does not exist');

    Write('press enter key to close..');
    Readln;
end.
```

이 예제를 실행하고 예제에 대한 텍스트 파일 이름을 입력하고 나면, 처음에는 CR/LF 값 (13/10)을 보게 되는데, 제각각의 문자 코드 (ASCII)를 표시하기 때문입니다. 리눅스에서는 10진수로 10의 값을 갖는 라인 피드 (LF)만 보게 됩니다.

그림이나 실행파일과 같은 다른 형식의 파일도 내부가 어떻게 생겼는지 보기 위해 표시할 수 있습니다.

이 예제에서 메모리에 97이라는 바이트 값으로 저장된 a 와 65라는 바이트 값으로 저장된 A와 같은 문자들의 숫자 값을 얻기 위해 *Chr* 함수를 사용했습니다.

1.46 날짜와 시간

날짜와 시간은 프로그래밍에 있어 가장 중요한 두 가지 요소입니다. 구매사항이나, 고지서 납부 등 트랜잭션에 대한 날짜와 시간을 저장할 필요가 있는 트랜잭션이나 어떤 처리 정보를 저장하는 프로그램에 중요합니다. 그런 다음에 이 프로그램들은 예를 들어 지난 달 또는 이번 달 동안 일어난 처리사항들과 트랜잭션을 결정할 수 있습니다.

프로그램들의 로그는 날짜/시간 기록과 관련된 일종의 처리 내용입니다. 어떤 프로그램이 언제 시작하고, 멈추고, 오류가 발생하거나 깨졌는지 알 필요가 있습니다.

*TDateTime*은 날짜/시간 정보를 저장할 때 사용할 수 있는 오브젝트 파스칼에 있는 형식입니다. 메모리에서 8바이트를 차지하는 배 정밀도 부동 소숫점입니다. 이 형식의 정밀도 부분은 시간을 표현하며, 전체적인 부분은 *1899년 12월 30일* 이후로 며칠이 지났는지에 대한 날짜를 표현합니다.

다음 예제를 통해 Now 함수를 이용하여 어떻게 현재의 날짜/시간 값을 표시하는지 보여드리겠습니다.

```
Program DateTime;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

begin
    Writeln('Current date and time: ', DateTimeToStr(Now));
    Write('Press enter key to close');
    Readln;
end.
```

Now 함수가 되돌려주는 *TDateTime* 형을 바꾸기 위해 *SysUtils* Unit에 들어있는 *DateTimeToStr* 함수를 사용했습니다.

이 변환을 사용하지 않으면, 실수 형태로 표시하는 인코딩한 날짜/시간을 얻게 됩니다.

```
    Writeln('Current date and time: ', Now);
```

또한 날짜 부분과 시간 부분만 표시해주는 두 가지의 날짜/시간 변환 함수도 있습니다.

```
    Writeln('Current date is: ', DateToStr(Now));
    Writeln('Current time is: ', TimeToStr(Now));
```

Date 함수는 오늘의 날짜 부분만 되돌려주고 Time 함수는 현재시간 부분만 되돌립니다.

```
    Writeln('Current date is: ', DateToStr(Date));
    Writeln('Current time is: ', TimeToStr(Time));
```

이 두 가지 함수들은 제각기 다른 부분에 0을 놓는데, 예를 들어 *Date* 함수는 현재 날짜를 되돌리면서 시간 부분에 0(시간 부분에서의 0은 12:00 am(자정) 을 의미)을 놓고, *Time* 함수는 현재 시스템 시간을 되돌리면서 날짜 부분에 0(날짜 부분에서의 0은 1899년 12월 30일을 의미)을 놓습니다.

이 사실은 *DateTimeToStr* 함수를 통해 확인해볼 수 있습니다.

```
    Writeln('Current date is: ', DateTimeToStr(Date));
    Writeln('Current time is: ', DateTimeToStr(Time));
```

*DateTimeToStr*은 컴퓨터의 날짜/시간 설정에 따라 날짜/시간을 보여줍니다. 이 결과는 두 가지의 컴퓨터 시스템 사이에서 변동되겠지만, *FormatDateTime* 함수라면 컴퓨터 설정과는 상관 없이 프로그래머가 작성한 형식대로 날짜와 시간을 표시할 것입니다.

```
    Writeln('Current date is: ',
        FormatDateTime('yyyy-mm-dd hh:mm:ss', Date));
    Writeln('Current time is: ',
        FormatDateTime('yyyy-mm-dd hh:mm:ss', Time));
```

다음 예제에서는 날짜/시간을 실수형태로 다룰 것이며, 이 값들에 대한 값을 더하고 빼보겠습니다.

```
begin
    Writeln('Current date and time is ',
        FormatDateTime('yyyy-mm-dd hh:nn:ss', Now));
    Writeln('Yesterday time is ',
        FormatDateTime('yyyy-mm-dd hh:nn:ss', Now - 1));
    Writeln('Tomorrow time is ',
        FormatDateTime('yyyy-mm-dd hh:nn:ss', Now + 1));
    Writeln('Today + 12 hours is ',
        FormatDateTime('yyyy-mm-dd hh:nn:ss', Now + 1/2));
    Writeln('Today + 6 hours is ',
        FormatDateTime('yyyy-mm-dd hh:nn:ss', Now + 1/4));
    Write('Press enter key to close');
    Readln;
end.
```

날짜로부터 하나를 더하거나 빼면, 하루를 완전히 더하거나 뺄 것입니다. $\frac{1}{2}$ 이나 0.5와 같은 것을 더한다면, 반나절(12 시간)을 더하게 됩니다.

다음 예제에서는 연, 월, 일을 사용하여 날짜를 만들어보겠습니다.

```
var
    ADate: TDateTime;
begin
    ADate:= EncodeDate(1975, 11, 16);

    Writeln('My date of birth is: ', FormatDateTime('yyyy-mm-dd', ADate));
    Write('Press enter key to close');
    Readln;
end.
```

EncodeDate 함수는 연, 월, 일을 입력으로 받아들이며, *TDateTime* 형의 하나의 값으로 연/월/일 값을 되돌립니다.

EncodeTime 함수는 시, 분, 초 그리고 밀리초를 받아들이며, 하나의 *TDateTime* 값으로 시간 값을 되돌립니다.

```
var
    ATime: TDateTime;
begin
    ATime:= EncodeTime(19, 22, 50, 0);
    Writeln('Almughrib prayer time is: ', FormatDateTime('hh:mm:ss', ATime));
    Write('Press enter key to close');
    Readln;
end.
```

1.47 날짜/시간 비교하기

실수를 비교하는 것과 마찬가지로 두개의 날짜/시간 변수를 비교할 수 있습니다. 예를 들어 실수에서 9.3은 5.1보다 크며 *TDateTime* 값에 대해서도 마찬가지 입니다. 내일을 나타내는 Now + 1은 오늘(Now) 보다는 크며, 한 시간 이후를 나타내는 Now + 1/24도 역시 지금으로부터 두 시간 전을 나타내는 Now-2/24 보다 큽니다.

다음 예제에서 2012년 1월 1일 날짜를 변수에 넣고, 현재 날짜와 비교하여 이 날짜가 지났는지 아직 안 지났는지를 확인해보겠습니다.

```
var
    Year2012: TDateTime;
begin
    Year2012:= EncodeDate(2012, 1, 1);
    if Now < Year2012 then
        Writeln('Year 2012 is not coming yet')
    else
        Writeln('Year 2012 is already passed');
    Write('Press enter key to close');
    Readln;
end.
```

이 예제에 해당 날짜로부터 얼마나 남았는지 또는 지났는지 표시하는 새로운 함수를 넣을 수 있습니다.

```
var
    Year2012: TDateTime;
    Diff: Double;
begin
    Year2012:= EncodeDate(2012, 1, 1);
    Diff:= Abs(Now – Year2012);

    if Now < Year2012 then
    Writeln('Year 2012 is not coming yet, there are ',
        Format('%0.2f', [Diff]), ' days Remaining ')
    else
    Writeln('First day of January 2012 is passed by ',
        Format('%0.2f', [Diff]), ' Days');
    Write('Press enter key to close');
    Readln;
end.
```

*Diff*는 현재 날짜와 2012년 날짜 사이의 차이를 지닐 실수 변수입니다. 숫자의 절대 값(음의 부호가 없는 숫자)을 되돌려주는 *Abs* 함수도 사용하였습니다.

1.48 뉴스 기록 프로그램

이 예제에서는 뉴스 제목을 저장하기 위한 텍스트 파일을 사용할 것이고, 이에 덧붙여서 날짜와 시간을 함께 저장할 것입니다.

프로그램을 닫고 다시 열면, 먼저 입력한 뉴스 제목과 날짜/시간이 표시될 것입니다.

```
Program news;

{$mode objfpc}{$H+}
uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes , SysUtils
    { you can add units after this };

var
    Title: string;
    F: TextFile;
begin
    AssignFile(F, 'news.txt');
    if FileExists('news.txt') then
    begin
        // Display old news
        Reset(F);
        while not Eof(F) do
        begin
            Readln(F, Title);
            Writeln(Title);
        end;
        CloseFile(F);       // reading is finished from old news
        Append(F);          // open file again for appending
    end
    else
        Rewrite(F);

    Write('Input current hour news title: ');
    Readln(Title);
    Writeln(F, DateTimeToStr(Now), ', ', Title);
    CloseFile(F);

    Write('Press enter to close');
    Readln;
end.
```

1.49 상수

상수는 변수와 유사합니다. 이름을 가지며 값을 지니지만, 변수와는 달리 값을 수정하는 점이 다릅니다. 프로그램을 실행하는 동안에는 상수 값을 바꿀 수 없으며, 프로그램을 컴파일하기 전에 값이 결정됩니다.

이름을 부여하지 않고, 아래 예제와 같이 정수 값이나 문자열로 바로 사용하는 다른 방식을 통해 상수를 사용해왔습니다.

```
    Writeln(5);
    Writeln('Hello');
```

5 값은 프로그램을 실행하는 동안에는 바뀌지 않을 것입니다. 'Hello' 또한 문자열 상수입니다.

다음 예제에서와 같이 프로그램의 uses 절 다음에서 *Const* 키워드를 사용하여 상수를 정의할 수 있습니다.

1.50 연료 소비 프로그램

```
Program FuelConsumption;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes , SysUtils
    { you can add units after this };

const GallonPrice = 6.5;

var
    Payment: Integer;
    Consumption: Integer;
    Kilos: Single;
begin
    Write('How much did you pay for your car"s fuel: ');
    Readln(Payment);
    Write('What is the consumption of your car? (Kilos per Gallon): ');
    Readln(Consumption);

    Kilos:= (Payment / GallonPrice) * Consumption;
```

```
    Writeln('This fuel will keep your car running for : ',
        Format('%0.1f', [Kilos]), ' Kilometers');
    Write('Press enter');
    Readln; end.
```

앞의 예제에서 다음 변수에 따라 현재 연료로 차가 달릴 수 있는 거리를 킬로미터 단위로 계산할 것입니다.

1. 차 연료 소모: 현재 차에 대한 갤런당 킬로미터 단위 거리를 저장하기 위해 *Consumption* 변수를 사용했습니다.
2. 연료: 현재 연료에 대해 얼마나 지불해야 하는지에 대한 금액을 저장하기 위해 *Payment* 변수를 사용했습니다.
3. 연료 비용: 현재 국가에서의 갤런 단위의 연료에 대한 가격을 저장하기 위해 *GallonPrice* 상수를 사용했습니다. 이 값은 사용자가 입력할 것이 아니고, 대신에 프로그래머가 정의할 것입니다.

프로그램에서 자주 사용하는 같은 값을 사용한다면 상수를 권장합니다. 이 값을 바꿀 필요가 있다면, 코드의 앞부분에서 한 번에 수정할 수 있습니다.

1.51 서수형

서수형은 숫자 대신 단어 표시를 사용하는 정수값입니다. 예를 들어, 언어 변수(Arabic / English / French)를 정의하려 한다면, Arabic에 1을, English에 2를, French에 3을 사용할 수 있습니다. 다른 프로그래머들은 이 값에 대한 주석을 찾기 전에는 1과 2, 3에 대한 값을 알지 못합니다. 다음 예제와 같이 서수형을 사용한다면, 좀 더 알아보기 쉬울 것입니다.

```
Program OrdinalTypes;

{$mode objfpc}{$H+}
uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

type
    TLanguageType = (ltArabic, ltEnglish);
var
    Lang: TLanguageType;
    AName: string;
    Selection: Byte;
begin
    Write('Please select Language: 1 (Arabic), 2 (English)');
    Readln(Selection);
    if Selection = 1 then
        Lang:= ltArabic
    else
    if selection = 2 then
        Lang:= ltEnglish
    else
        Writeln('Wrong entry');
    if Lang = ltArabic then

        Write('ماهو اسمك:  ');
    else
        if Lang = ltEnglish then
            Write('What is your name: ');
    Readln(AName);
    if Lang = ltArabic then
    begin

        Writeln('مرحبً بك', AName);

        Write('الرجاء الضغط على مفتاح إدخال لغلق البرنامج');
    end
    else
    if Lang = ltEnglish then
```

```
    begin
        Writeln('Hello ', AName);
        Write('Please press enter key to close');
    end;
    Readln;
end.
```

실수와 문자열은 서수형이 아닌 반면에 정수, 문자, 부울린은 서수형입니다.

1.52 Set

Set 형식은 하나의 변수에 여러가지 속성이나 특징을 지닐 수 있습니다. Set은 서수 값으로만 사용합니다.

예를 들어, 프로그램에 운영체제 지원에 대해 정의하려 한다면, 다음처럼 할 수 있습니다.

1. 운영 체제를 나타내는 서수형을 정의합니다: TApplicationEnv

```
TApplicationEnv = (aeLinux, aeMac, aeWindows);
```

2. 프로그램을 TApplicationEnv의 Set으로 정의합니다.
예를 들면

```
FireFox: set of TApplicationEnv;
```

3. 프로그램 Set 변수에 운영체제 값을 넣습니다.

```
FireFox := [aeLinux, aeWindows];
```

```
Program Sets;
{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

type
    TApplicationEnv = (aeLinux, aeMac, aeWindows);
var
    FireFox: set of TApplicationEnv;
    SuperTux: set of TApplicationEnv;
    Delphi: set of TApplicationEnv;
    Lazarus: set of TApplicationEnv;
begin
    FireFox:= [aeLinux, aeWindows];
    SuperTux:= [aeLinux];
    Delphi:= [aeWindows];
    Lazarus:= [aeLinux, aeMac, aeWindows];
```

```
    if aeLinux in Lazarus then
        Writeln('There is a version for Lazarus under Linux')
    else
        Writeln('There is no version of Lazarus under linux');

    if aeLinux in SuperTux then
        Writeln('There is a version for SuperTux under Linux')
    else
        Writeln('There is no version of SuperTux under linux');

    if aeMac in SuperTux then
        Writeln('There is a version for SuperTux under Mac')
    else
        Writeln('There is no version of SuperTux under Mac');
    Readln;
end.
```

또한 정수

```
    if Month in [1,3,5,7,8,10,12] then
        Writeln('This month contains 31 days');
```

또는 문자와 같은 다른 서수형에 대한 Set 구문을 사용할 수 있습니다.

```
    if Char in ['a', 'A'] then
        Writeln('This letter is A');
```

1.53 예외 처리

오류에는 두가지 종류가 있습니다. 하나는 var 섹션에서 정의하지 않은 변수를 사용하거나 잘못된 문법으로 구문을 작성해서 일어나는 컴파일 오류입니다. 이 형태의 오류들은 프로그램이 컴파일 되는 것을 막으며, 컴파일러는 적절한 메시지를 표시하고 오류가 있는 줄을 가리킵니다.

두번째 오류는 실행시간 오류입니다. 이 유형의 오류는 예를 들어 다음과 같은 상황에서 0으로 나누었을 때, 프로그램 실행 중에 일어납니다.

```
x := y / z;
```

이는 유효한 구문이지만, 실행 중 사용자가 z 변수에 0 값을 입력할 수 있으며, 프로그램은 깨질 것이고, *0으로 나눔(Division by zero)* 오류 메시지를 표시할 것입니다.

존재하지 않는 파일을 열려고 하거나(*File not found*) 읽기 전용 디렉터리에 파일을 만들려고 하는 것 또한 실행 시간 오류를 만들어냅니다.

컴파일러는 프로그램을 실행한 다음에만 일어나는 몇몇 오류는 잡을 수 없습니다.

실행 시간 오류로 충돌이 일어나지 않을 믿을 수 있는 프로그램을 만들기 위해 예외 처리를 사용할 것입니다.

오브젝트 파스칼에서는 예외 처리를 하는 여러가지 방법이 있습니다.

1.54 try except 구문

구문:

```
try
    // Start of protected code
    CallProc1;
    CallProc2;
    // End of protected code
except
on e: exception do //Exception handling
begin
    Writeln('Error:' + e.message);
end;
end;
```

다음은 나누기에 대한 예제입니다.

```
Program ExceptionHandling;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    x, y: Integer;
    Res: Double;
begin
    try
        Write('Input x: ');
        Readln(x);
        Write('Input y: ');
        Readln(y);
        Res:= x / y;
        Writeln('x / y = ', Res);

    except
    on e: exception do
    begin
        Writeln('An error occurred: ', e.message);
    end;
    end;
    Write('Press enter key to close');
    Readln;
end.
```

try 구문에 두개의 섹션이 있습니다. 처음 부분은 *try...except* 사이에 있는 보호할 필요가 있는 블록이고, 다른 부분은 *except .. end* 사이에 있습니다.

처음 섹션(*try except*)에서 무엇인가가 잘못되었다면, 프로그램은 *except* 섹션 (*except..end*)으로 갈 것이고 충돌이 일어나지 않겠지만, 적당한 오류 메시지를 표시하고 실행을 계속할 것입니다.

y 값이 0이라면 예외가 발생할 수 있는 줄입니다.

```
    Res := x / y
```

예외가 발생하지 않았다면, *except .. end* 부분을 실행하지 않을 것입니다.

1.55 try finally

구문:

```
try
    // Start of protected code
    CallProc1;
    CallProc2;
    // End of protected code
finally
    Writeln('This line will be printed in screen for sure');
end;
```

try finally 방식을 사용한 나누기 프로그램입니다.

```
Program ExceptionHandling;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

var
    x, y: Integer;
    Res: Double;
begin
    try
        Write('Input x: ');
        Readln(x);
        Write('Input y: ');
        Readln(y);
        Res:= x / y;
        Writeln('x / y = ', Res);
    finally
        Write('Press enter key to close');
        Readln;
    end;
end.
```

이제는 오류 발생 여부와는 관계없이 모든 경우에 *finally .. end* 부분을 실행할 것입니다.

1.56 예외 일으키기

때로는 논리 오류를 막기 위해 예외를 만들고 일으킬 필요가 있습니다. 예를 들어 사용자가 월 수를 13으로 입력 했다면, 월 수의 범위를 벗어났다는 것을 알리는 예외를 일으킬 수 있습니다.

```
Program RaiseExcept;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

var
    x: Integer;
begin
    Write('Please input a number from 1 to 10: ');
    Readln(x);
    try

        if ( x < 1 ) or ( x > 10 ) then // raise exception
            raise exception.Create('x is out of range');
        Writeln(' X * 10 = ', x * 10);

    except
    on e: exception do // catch my exception
    begin
        Writeln('Error: ' + e.Message);
    end;
    end;
    Write('Press enter to close');
    Readln;

end.
```

사용자가 1부터 10까지의 범위 밖의 값을 입력하면, 예외(*X is out of range*)가 만들어질 것이고 이 부분에서 예외 처리가 없다면, 프로그램은 문제에 충돌할 것입니다. 왜냐하면 *try except* 사이에 *raise* 키워드를 포함한 코드를 작성했고, 프로그램은 충돌하지 않겠지만, 대신 오류 메시지를 표시할 것이기 때문입니다.

Chapter 2

구조적 프로그래밍

2.1 도입

구조적 프로그래밍은 프로그램이 커진 이후에 나타났습니다. 하나의 파일에 구조화하지 않은 코드를 사용할 때 큰 프로그램은 점점 알아볼 수 없게 되었고, 유지할 수 없게 되었습니다.

구조적 프로그래밍에서는 프로그램의 소스코드를 *프로시저*와 *함수*라는 더 작은 부분으로 나눌 수 있으며, *Unit*이라고 하는 각각의 코드 파일의 주제와 관련된 *프로시저*와 *함수*들을 합칠 수 있습니다.

구조적 프로그래밍은 다음과 같은 장점이 있습니다.

1. 모듈과 프로시저를 알아볼 수 있게 프로그램의 코드를 **분할**합니다.

2. 코드 **재 사용성**: 프로시저와 함수는 코드를 재작성하거나 복사할 필요 없이 임의의 코드 일부에서 호출할 수 있습니다.

3. 프로그래머는 동시에 하나의 프로젝트 안에서 **공유**하고 **참여**할 수 있습니다. 프로그래머 각자는 그들 자신의 프로시저와 함수를 나누어서 작성할 수 있고, 이것들을 프로젝트에 통합할 수 있습니다.

4. 프로그램의 **관리**와 **개선**이 쉬워집니다. 프로시저 안에서 버그를 쉽게 찾을 수 있으며, 프로시저와 함수의 개선, 새로운 프로시저 및 함수 작성, 새로운 Unit 추가 등을 쉽게할 수 있습니다.

5. **모듈** 및 **계층**의 도입: 프로그램을 제각각의 논리적인 계층과 모듈로 나눌 수 있습니다. 예를 들어 데이터를 파일로부터 읽거나 파일로 기록하는 프로시저를 포함한 Unit과 규칙을 표현하고 계층을 검증하는 다른 Unit, 그리고 세번째로 사용자 인터페이스 계층을 작성할 수 있습니다.

2.2 프로시저

이전 장에서 *Writeln*, *Readln*, *Reset* 등과 같은 프로시저를 이미 사용했지만, 이번에는 우리 프로그램에서 사용할 *프로시저*를 작성하고자 합니다.

다음 예제에서는 *SayHello*와 *SayGoodbye* 두 개의 프로시저를 작성했습니다.

```
Program Structured;
{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

procesure SayHello;
begin
    Writeln('Hello there');
end;

procedure SayGoodbye;
begin
    Writeln('Good bye');
end;

begin // Here main application starts
    Writeln('This is the main application started');
    SayHello;
    Writeln('This is structured application');
    SayGoodbye;
    Write('Press enter key to close');
    Readln;
end.
```

작은 프로그램으로서 *begin .. end*를 지닌 프로시저와 메인 프로그램 코드에서 이것을 호출하는 것을 보았습니다.

2.3 인자

다음 예제에서는 프로시저를 호출할 때 변수를 전달하는 *인자*를 소개합니다.

```
procedule WriteSumm(x, y:Integer);
begin
    Wrilteln('The summation of ', x, ' + ', y , ' = ', x + y)
end;

begin
    WriteSumm(2,7);
    Write('Press enter key to close');
    Readln;
end.
```

메인 프로그램에서 *WriteSumm* 프로시저를 호출했고 값 *2*와 *7*을 프로시저에 전

달했으며, 프로시저는 이 값들에 대한 합계 결과를 화면에 출력하기 위해 x, y 정수형 변수로 받을 것입니다.

다음 예제에서는 프로시저를 사용하여 음식점 프로그램을 다시 작성했습니다.

2.4 프로시저를 사용한 음식점 프로그램

```
procedure Menu;
begin
    Writeln('Welcome to Pascal Restaurant. Please select your order');
    Writeln('1 – Chicken          (10$)');
    Writeln('2 – Fish             (7$)');
    Writeln('3 – Meat             (8$)');
    Writeln('4 – Salad            (2$)');
    Writeln('5 - Orange Juice     (1$)');
    Writeln('6 – Milk             (1$)');
    Writeln;
end;

procedure GetOrder(AName: string; Minutes: Integer);
begin
    Writeln('You have ordered : ', AName, ', this will take ',
        Minutes, ' minutes');
end;

// Main application
var
    Meal: Byte;
begin
    Menu;
    Write('Please enter your selection: ');
    Readln(Meal);

    case Meal of
        1: GetOrder('Chicken', 15);
        2: GetOrder('Fish', 12);
        3: GetOrder('Meat', 18);
        4: GetOrder('Salad', 5);
        5: GetOrder('Orange juice', 2);
        6: GetOrder('Milk', 1);
    else
        Writeln('Wrong entry');
    end;
    Write('Press enter key to close');
    Readln;
end.
```

이제 메인 프로그램은 더 작아지고 좀 더 잘 알아볼 수 있게 되었습니다. 다른 부분에 대한 세부 내용은 *주문 받기*나 *메뉴 표시하기*와 같은 프로시저로 분리했습니다.

2.5 함수

함수는 프로시저와 유사하지만 값을 되돌리는 기능도 있습니다. 우리는 이전에 텍스트를 대문자로 바꿔주는 *UpperCase*와 숫자의 절대값을 되돌려주는 *Abs*와 같은 함수를 사용했습니다.

다음 예제에서는 두 개의 정수값을 받아서 값들의 합을 되돌려주는 *GetSumm* 함수를 작성했습니다.

```
function GetSumm(x, y: Integer): Integer;
begin
    Result := x + y;
end;

var
    Sum: Integer;
begin
    Sum := GetSumm(2, 7);
    Writeln('Summation of 2 + 7 = ', Sum);
    Write('Press enter key to close');
    Readln;
end.
```

함수를 정수형으로 선언하고, 함수의 되돌릴 값을 나타내기 위해 *Result* 키워드를 사용했음을 살펴보도록 합니다.

메인 프로그램에서 함수의 값을 받기 위해 *Sum*변수를 사용했지만, 이 중간 변수를 없애고 *Writeln* 프로시저에서 이 함수를 호출할 수 있습니다. 이것이 함수와 프로시저의 차이점 중 하나입니다. 다른 프로시저나 함수에서 함수를 입력 인자로써 호출할 수 있지만, 프로시저는 다른 함수와 프로시저의 인자로서 호출할 수 없습니다.

```
function GetSumm(x, y: Integer): Integer;
begin
    Result := x + y;
end;

begin
    Writeln('Summation of 2 + 7 = ', GetSumm(2, 7));
    Write('Press enter key to close');
    Readln; end.
```

다음 예제에서는 함수를 사용하여 음식점 프로그램을 다시 작성했습니다.

2.6 함수를 사용한 음식점 프로그램

```
procedure Menu;
begin
    Writeln('Welcome to Pascal Restaurant. Please select your order');
    Writeln('1 – Chicken          (10$)');
    Writeln('2 – Fish             (7$)');
    Writeln('3 – Meat             (8$)');
    Writeln('4 – Salad            (2$)');
    Writeln('5 - Orange Juice     (1$)');
    Writeln('6 – Milk             (1$)');
    Writeln('X - nothing');
    Writeln;
end;

function GetOrder(AName: string; Minutes, Price: Integer): Integer;
begin
    Writeln('You have ordered : ', AName, ', this will take ',
        Minutes, ' minutes');
    Result := Price;
end;

// Main application
var
    Selection: Char;
    Price: Integer;
    Total: Integer;
begin
    Total := 0;
    repeat
        Menu;
        Write('Please enter your selection: ');
        Readln(Selection);

        case Selection of
            '1': GetOrder('Chicken', 15, 10);
            '2': GetOrder('Fish', 12, 7);
            '3': GetOrder('Meat', 18, 8);
            '4': GetOrder('Salad', 5, 2);
            '5': GetOrder('Orange juice', 2, 1);
            '6': GetOrder('Milk', 1, 1);
            'x', 'X': Writeln('Thanks');
            else
            begin
                Writeln('Wrong entry');
                Price := 0;
            end;
        end;
```

```
            Total := Total + Price;

        until (Selection = 'x') or (Selection = 'X');
        Writeln('Total price                =', Total);
        Write('Press enter key to close');
        Readln;
end.
```

2.7 지역 변수

변수들을 정의할 때, 프로시저나 함수 안에서만 사용할 수 있도록 지역적으로 정의할 수 있습니다. 이 변수들은 메인 프로그램 코드나 다른 프로시저 또는 함수에서 접근할 수 없습니다.

예제:

```
procedure Loop(Counter: Integer);
var
    i: Integer;
    Sum: Integer;
begin
    Sum := 0;
    for i := 1 to Counter do
        Sum := Sum + i;
    Writeln('Summation of ', Counter, ' number is: ', Sum);
end;

begin // Main program section
    Loop;
    Write('Press enter key to close');
    Readln;
end.
```

Loop 프로시저에는 *Sum*과 i 두 지역변수가 있습니다. 지역 변수는 프로시저의 실행이 끝나기 전까지 임시로 변수를 할당하는 메모리의 일부인 스택 메모리에 저장합니다. 이는 프로그램 실행이 코드의 이 줄에 도달할 때 지역변수에 접근할 수 없으며, 지역변수를 덮어쓸 수 있다는 것을 의미합니다.

```
    Write('Press enter key to continue');
```

전역 변수들은 메인 프로그램과 다른 프로시저 및 함수에서 접근할 수 있습니다. 이들은 프로그램이 끝날 때까지 값을 지닐 수 있습니다. 하지만, 프로그램의 구조를 깰 수 있으며, 오류를 추적하기 어렵게 만드는데, 우리가 전역 변수를 초기화 하는 것을 잊었을 때, 어떤 프로시저는 알 수 없는 값을 넣고 잘못된 행동을 취하도록 전역 변수 값을 바꿀 수 있기 때문입니다.

지역 변수를 정의하는 것은 전역 변수 값에 대한 걱정 없이, 프로시저와 함수를 어디든 이식하고 호출할 수 있도록 돕는 개별성을 보장합니다.

2.8 뉴스 데이터베이스 프로그램

이 예제에서는 뉴스 *제목을 추가*하고, 모든 뉴스를 *표시*하며, *검색*하는 세 개의 프로시저와, 사용자가 실행하고자 하는 기능을 사용자에게 선택하도록 *메뉴*를 *표시*하는 하나의 함수가 있습니다.

```
Program news;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this };

type
    TNews = record
        ATime: TDateTime;
        Title: string[100];
    end;
procedure AddTitle;
var
    F: file of TNews;
    News: TNews;
begin
    AssignFile(F, 'news.dat');
    Write('Input current news title: ');
    Readln(News.Title);
    News.ATime:= Now;
    if FileExists('news.dat') then
    begin
        FileMode:= 2; // Read/Write
        Reset(F);
        Seek(F, System.FileSize(F)); // Go to last record to append
    end
    else
        Rewrite(F);
    Write(F, News);
    CloseFile(F);
end;

procedure ReadAllNews;
var
    F: file of TNews;
    News: TNews;
```

```
begin
    AssignFile(F, 'news.dat');
    if FileExists('news.dat') then
    begin
        Reset(F);
        while not Eof(F) do
        begin
            Read(F, News);
            Writeln('-------------------------');
            Writeln('Title: ', News.Title);
            Writeln('Time : ', DateTimeToStr(News.ATime));
        end;
    CloseFile(F);
    end
    else
        Writeln('Empty database');
end;

procedure Search;
var
    F: file of TNews;
    News: TNews;
    Keyword: string;
    Found: Boolean;
begin
    AssignFile(F, 'news.dat');
    Write('Input keyword to search for: ');
    Readln(Keyword);
    Found:= False;
    if FileExists('news.dat') then
    begin
        Reset(F);
        while not Eof(F) do
        begin
            Read(F, News);
            if Pos(LowerCase(Keyword), LowerCase(News.Title)) > 0 then
            begin
                Found:= True;
                Writeln('-------------------------');
                Writeln('Title: ', News.Title);
                Writeln('Time : ', DateTimeToStr(News.ATime));
            end;
        end;
        CloseFile(F);
        if not Found then
            Writeln(Keyword, ' Not found');
    end
```

```
    else
        Writeln('Empty database');
end;

function Menu: char;
begin
    Writeln;
    Writeln('.....News database.....');
    Writeln('1. Add news title');
    Writeln('2. Display all news');
    Writeln('3. Search');
    Writeln('x. Exit');
    Write('Please input a selection : ');
    Readln(Result);
end;

// Main application
var
    Selection: Char;
begin
    repeat
        Selection:= Menu;
        case Selection of
        '1': AddTitle;
        '2': ReadAllNews;
        '3': Search;
        end;
    until LowerCase(Selection) = 'x';
end.
```

이 프로그램은 알아보기 쉬우며 메인 섹션에는 작고 깔끔한 코드를 지니고 있습니다. 다른 부분에 영향을 주거나 수정하지 않고도 임의의 프로시저에 새로운 기능을 추가할 수 있습니다.

2.9 입력 인자로서의 함수

앞서 이야기 한 바와 같이, 함수는 프로시저 혹은 함수의 입력 인자로서 호출할 수 있는데, 값과 같은 존재로 다룰 수 있기 때문입니다.

아래 예제를 보도록 하겠습니다.

```
function DoubleNumber(x: Integer): Integer;
begin
    Result := x * 2;
end;

// Main

begin
    Writeln('The double of 5 is : ', DoubleNumber(5));
    Readln;
end.
```

참고로 *Writeln* 프로시저에서 *DoubleNumber* 함수를 호출했습니다.

다음 수정한 예제에서는 함수의 결과를 저장하기 위해 중간 변수를 사용했고, *Writeln* 프로시저에 사용한 중간 변수를 입력으로 사용했습니다.

```
function DoubleNumber(x: Integer): Integer;
begin
    Result := x * 2;
end;

// Main

var
    MyNum: Integer;
begin
    MyNum:= DoubleNumber(5);
    Writeln('The double of 5 is : ', MyNum);
    Readln;
end.
```

또한 if 조건문과 순환문의 조건부에서 함수를 호출할 수 있습니다.

```
function DoubleNumber(x: Integer): Integer;
begin
    Result := x * 2;
end;

// Main

var
    MyNum: Integer;
begin
    if DoubleNumber(5) > 10 then
        Writeln('This number is larger than 10')
    else
        Writeln('This number is equal or less than 10');
    Readln;
end.
```

2.10 프로시저와 함수의 출력 인자

이전 함수 사용에서 함수의 결과로서 하나의 값만 되돌려줄 수 있다는 것을 알았는데, 어떻게 하면 함수나 프로시저에서 하나 이상의 값을 되돌려 줄 수 있을까요?

그럼 실험을 해봅시다.

```
procedure DoSomething(x: Integer);
begin
    x := x * 2;
    Writeln('From inside procedure: x = ', x);
end;

// main

var
    MyNumber: Integer;
begin
    MyNumber:= 5;

    DoSomething(MyNumber);
    Writeln('From main program, MyNumber = ', MyNumber);
    Writeln('Press enter key to close');
    Readln;
end.
```

이 예제에서는 *DoSomething* 프로시저가 x를 정수 값으로 받은 다음에 2를 곱하고 이 값을 표시합니다.

프로그램의 메인 부분에서는 *MyNumber*를 정수형으로 선언했고, *5*라는 값을 넣었으며, *DoSomething* 프로시저에 인자로 전달했습니다. 이 경우 *MyNumber*의 값(*5*)은 x변수에 복사될 것입니다.

함수를 호출하고 나면 x의 값은 10이 되겠지만, 함수를 호출한 다음에 *MyNumber*의 값을 표시할 때는 여전히 *5*를 지니고 있는 것을 알게 될 것입니다. 이는 *MyNumber*와 x는 일반적으로 서로 다른 메모리 공간에 위치하고 있다는 뜻입니다.

MyNumber 원본 인자에 영향을 주지 않는 이러한 인자 전달의 유형을 *값에 의한 호출*이라고 부릅니다. 이런 값을 전달하기 위해 상수 값을 사용할 수도 있습니다.

```
DoSomething(5)
```

2.11 참조에 의한 호출

*DoSomething*의 *x*인자 선언부에 *var* 키워드를 추가하면 *x* 인자 값이 달라질 것입니다.

```
procedure DoSomething(var x: Integer);
begin
    x:= x * 2;
    Writeln('From inside procedure: x = ', x);
end;

// main

var
    MyNumber: Integer;
begin
    MyNumber:= 5;

    DoSomething(MyNumber);
    Writeln('From main program, MyNumber = ', MyNumber);
    Writeln('Press enter key to close');
    Readln;
end.
```

이제 *x*에 따라 *MyNumber*의 값이 달라질 것이며, 이는 이들 변수가 같은 메모리 위치를 공유한다는 의미입니다.

이 때, 우리는 (상수가 아니라) 같은 형의 변수를 프로시저로 전달해야 합니다. 만약 인자를 Byte로 선언했다면, *MyNumber*는 Byte로 선언해야 합니다. 반면에 인자가 정수형이면 *MyNumber*역시 정수형으로 선언해야 합니다.

다음 예제에서 인자에 변수를 필요로 하는 *DoSomething*을 호출했을 때, 오류가 생길 것입니다.

```
    DoSomething(5);
```

값에 의한 호출에서는 이전 코드에서는 사용할 수 있었는데, 인자로 전달한 값에 대해서만 다루었고, 5가 값이었습니다. 하지만 *참조에 의한* 호출에서는 인자로 전달한 변수에 대해 취급하며, 전달한 변수의 값에 대해 동작합니다.

다음 예제에서는 두 개의 변수를 전달할 것이고, 프로시저에서 두 변수의 값을 바꿀 것입니다.

```
procedure SwapNumbers(var x, y: Integer);
var
    Temp: Integer;
begin
    Temp:= x;
    x:= y;
    y:= Temp;
end;

// main

var
    A, B: Integer;
begin

    Write('Please input value for A: ');
    Readln(A);

    Write('Please input value for B: ');
    Readln(B);

    SwapNumbers(A, B);
    Writeln('A = ', A, ', and B = ', B);
    Writeln('Press enter key to close');
    Readln;
end.
```

2.12 Unit

파스칼의 Unit은 수많은 프로그램에서 사용할 수 있는 프로시저, 함수, 상수, 사용자 정의 형을 가진 라이브러리입니다.

Unit은 다음 목적을 위해 사용합니다.

1. 외부 Unit의 프로그램에서 자주 사용하는 프로시저나 함수를 모아놓습니다. 이는 소프트웨어 개발시 코드 재사용성 요건을 충족합니다.
2. 하나의 실체 안에서 제각각의 작업을 수행할 때 사용할 프로시저나 함수를 모아놓습니다. 관련성 없는 프로시저들로 메인 프로그램의 소스 코드의 규모를 불리는 대신에, Unit을 사용하여 논리적인 모듈로 프로그램을 나누는 것이 더 좋습니다.

새 Unit을 만들기 위해 라자루스 메뉴의 File/New Unit으로 이동하면, 라자루스는 다음과 같은 양식을 만듭니다.

```
unit Unit1;

    {$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils;

implementation

end.
```

새 Unit을 만들고 나면, *Test*와 같은 정해진 이름을 사용하여 저장할 것입니다. 이 소스 코드를 *Test.pas*라는 이름의 파일로 저장하겠지만, Unit의 이름인 *Test*는 프로그램에 그대로 남을 것입니다.

이제 프로시저, 함수, 그리고 다른 재사용가능한 코드의 작성을 시작해볼 수 있습니다.

```
unit Unit1;

    {$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils;

const
    GallonPrice = 6.5;

function GetKilometers(Payment, Consumption: Integer):Single;
```

```
implementation

function GetKilometers(Payment, Consumption: Integer):Single;
begin
    Result:= (Payment / GallonPrice) * Consumption;
end;

end.
```

GallonPrice 상수와 다른 프로그램에서 호출할 *GetKilometers* 함수를 작성했습니다.

또한 Unit의 외부에서 접근 가능하도록 하기 위해 Unit의 *interface* 부분에 함수의 머리 부분을 넣었습니다. 프로그램에서는 Unit의 *interface* 부분만 접근할 수 있습니다.

이 Unit을 사용하려면, Unit(Test.pas)과 같은 디렉터리에 새로운 프로그램을 만들고 uses절에 이 Unit을 추가합니다.

```
program PetrolConsumption;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils
    { you can add units after this }, Test;

var
    Payment: Integer;
    Consumption: Integer;
    Kilos: Single;
begin
    Write(How much did you pay for your car"s petrol: ');
    Readln(Payment);
    Write('What is the consumption of your car (Kilos per one Gallon) ');
    Readln(Consumption);

    Kilos:= GetKilometers(Payment , Consumption);

    Writeln('This petrol will keep your car running for: ',
        Format('%0.1f', [Kilos]), ' Kilometers');
    Write('Press enter');
    Readln;
end.
```

GetKilometers 함수로 이동하려면, *GetKilometers* 함수의 소스 코드를 표시하기 위해 *Ctrl* 키를 누른 상태에서 마우스로 이 이름에 가져다가 누를 수 있습니다. 라자루스나 델파이는 *Test* Unit을 바로 열고 해당 함수를 표시할 것입니다.

또한 Unit의 이름(*Test*)에 커서를 가져간 후 *Alt* + *Enter*를 눌러서 편집기에 *Test* Unit을 열 수 있습니다.

다음 같은 경우 프로그램에서 Unit에 접근할 수 있습니다.

1. 앞의 예제에서 했던 프로그램과 같이, 같은 디렉터리 안에 Unit 파일이 존재할 때
2. 라자루스에서 Unit을 열고 Project/Add Editor File to project 를 마우스로 눌러서 프로젝트에 Unit을 추가할 때
3. Project/Compiler Options/Compiler Options/Paths/Other Unit Files 에서 Unit의 경로를 추가할 때

2.13 라자루스와 프리 파스칼에서의 Unit

Unit은 라자루스와 프리 파스칼에서 가장 중요한 구성 요소입니다. 대부분의 프로시저와 함수, 형식, 클래스, 구성요소들을 Unit에서 볼 수 있습니다.

프리 파스칼은 여러가지 방식의 프로그램에서 사용하며 기능적으로 제공하는 풍부한 Unit의 모음을 지니고 있습니다. 이 덕분에 라자루스와 프리 파스칼을 사용하여 프로그램을 개발하는 것은 쉽고 빠릅니다.

일반적인 프로시저와 함수가 들어있는 프리 파스칼과 라자루스 Unit의 예로 *SysUtils*와 *Classes*가 있습니다.

2.14 프로그래머가 작성한 Unit

프로그래머들은 라자루스의 Unit을 사용할 수 있으며, 그들이 사용할 Unit을 손수 작성할 수 있습니다. 프로그래머들이 작성한 *Unit*은 그들이 특별히 필요로 하는 것을 나타냅니다. 예를 들어 개발자가 차고를 위한 소프트웨어를 작성한다면, 차 번호등을 사용하여 차를 추가하고, 차를 검색하기 위한 프로시저가 들어있는 Unit을 만들 수 있습니다.

Unit에 프로시저와 함수를 놓는 것은 한 사람 이상의 개발자들에게 알아보기 쉽고 개발하기에 용이하도록 하는데, 제 각각의 요소를 하나 이상의 모듈(*unit*)에 집중할 수 있고, 각자가 *Unit*을 기능 독립적으로 시험(*단위 테스트*)할 수 있으며, 심지어는 하나의 프로젝트에 이 *Unit*들을 통합할 수도 있기 때문입니다.

2.15 헤지라력(이슬람 달력)

헤지라력은 음력을 기반으로 하며, 무슬림들이 만들었습니다. 우리는 아래 사실을 근거로 하여 그레고리안력을 음(헤지라)력으로 변환하도록 도와줄 Unit을 만들려고 합니다.

1. 헤지라력의 첫 날은 그레고리안력의 622년 7월 16일입니다.
2. 헤지라의 1년은 354.367056일 입니다.
3. 헤지라의 한 달은 29.530588일 입니다.

헤지라력은 현재 달의 주기를 가져올 때 사용할 수 있습니다.

아래는 헤지라 Unit의 코드입니다.

```
{
*****************************************************************************

HejriUtils:      Hejri Calnder converter, for FreePascal and Delphi
Author:          Motaz Abdel Azeem
email:           motaz@code.sd
Home page:       http://motaz.freevar.com/
License:         LGPL
Created on:      26.Sept.2009
Last modifie:    26.Sept.2009

*****************************************************************************
}
unit HejriUtils;

{$IFDEF FPC}
{$mode objfpc}{$H+}
{$ENDIF}

interface
uses
    Classes, SysUtils;
const
    HejriMonthsEn: array [1 .. 12] of string = ('Moharram', 'Safar', 'Rabie Awal',
    'Rabie Thani', 'Jumada Awal', 'Jumada Thani', 'Rajab', 'Shaban',
    'Ramadan', 'Shawal', 'Thi-Alqaida', 'Thi-Elhajah');


    HejriMonthsAr: array [1 .. 12] of string = ( 'محرم', 'صفر', 'ربيع أول',

    'ربيع ثاني', 'جمادى الأول', 'جمادى الآخر', 'رجب', 'شعبان',
    'رمضان',

    'شوال', 'ذي القعدة', 'ذي الحجة' );
```

```
    HejriYearDays = 354.367056;
    HejriMonthDays = 29.530588;

    procedure DateToHejri(ADateTime: TDateTime; var Year, Month,
    Day: Word);
    function HejriToDate(Year, Month, Day: Word): TDateTime;

    procedure HejriDifference(Year1, Month1, Day1, Year2, Month2,
    Day2: Word; var YearD, MonthD, DayD: Word);
implementation

var
    HejriStart : TDateTime;
procedure DateToHejri(ADateTime: TDateTime;
                      var Year, Month, Day: Word);
var
    HejriY: Double;
    Days: Double;
    HejriMonth: Double;
    RDay: Double;
begin
    HejriY:= ((Trunc(ADateTime) - HejriStart - 1) / HejriYearDays);
    Days:= Frac(HejriY);
    Year:= Trunc(HejriY) + 1;
    HejriMonth:= ((Days * HejriYearDays) / HejriMonthDays);

    Month:= Trunc(HejriMonth) + 1;
    RDay:= (Frac(HejriMonth) * HejriMonthDays) + 1;
    Day:= Trunc(RDay);
end;

function HejriToDate(Year, Month, Day: Word): TDateTime;
begin
    Result:= (Year - 1) * HejriYearDays + (HejriStart - 0) +
        (Month - 1) * HejriMonthDays + Day + 1;
end;

procedure HejriDifference(Year1, Month1, Day1, Year2, Month2,
                      Day2: Word; var YearD, MonthD, DayD: Word);
var
    Days1: Double;
    Days2: Double;
    DDays: Double;
    RYear, RMonth: Double;
begin
    Days1:= Year1*HejriYearDays + (Month1 - 1)*HejriMonthDays + Day1 - 1;
    Days2:= Year2*HejriYearDays + (Month2 - 1)*HejriMonthDays + Day2 - 1;
    DDays:= Abs(Days2 – Days1);
```

```
    RYear:= DDays / HejriYearDays;
    RMonth:= (Frac(RYear) * HejriYearDays) / HejriMonthDays;
    DayD:= Round(Frac(RMonth) * HejriMonthDays);

    YearD:= Trunc(RYear);
    MonthD:= Trunc(RMonth);
end;
initialization

    HejriStart:= EncodeDate(622, 7, 16);

end.
```

HejriUtils Unit은 아래의 프로시저와 함수들이 있습니다.

1. *DateToHejri*: 이 프로시저는 *그레고리안 날짜*를 *헤지라 날짜*로 바꿔주는데 사용하며, 예제는 다음과 같습니다.

```
program Project1;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes , HejriUtils, SysUtils
    { you can add units after this };

var
    Year, Month, Day: Word;
begin
    DateToHejri(Now, Year, Month, Day);
    Writeln('Today in Hejri: ', Day, '-', HejriMonthsEn[Month],
        '-', Year);
    Readln;
end.
```

2. *HejriToDate*: 이 함수는 *헤지라 날짜*를 그레고리안 *TDateTime* 값으로 바꿔주는데 사용합니다.
3. *HejriDifference*: 이 프로시저는 두 *헤지라 날짜*의 *연*, *월*, *일* 차이를 계산하는데 사용합니다.

2.16 프로시저와 함수 오버로딩

오버로딩은 하나 이상의 프로시저나 함수를 이름은 같지만 인자를 다르게 하여 작성할 수 있음을 의미합니다. 다른 인자라는 것은 인자 형의 다름이나 인자 수의 다름을 의미합니다.

예를 들어 Sum 함수의 두 가지 버전을 작성하려고 합니다. 하나는 받아들이는 인자들과 되돌리는 값이 정수형이고, 다른 하나는 받아들이는 인자들과 되돌리는 값이 실수형입니다.

```
program sum;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

function Sum(x, y: Integer): Integer; overload;
begin
    Result:= x + y;
end;

function Sum(x, y: Double): Double; overload;
begin
    Result:= x + y;
end;

var
    j, i: Integer;
    h, g: Double;
begin
    j:= 15;
    i:= 20;
    Writeln(j, ' + ', i, ' = ', Sum(j, i));
    h:= 2.5;
    g:= 7.12;
    Writeln(h, ' + ', g, ' = ', Sum(h, g));
    Write('Press enter key to close');
    Readln;
end.
```

"이 함수는 오버로드 되었습니다"라는 의미를 지닌 예약어 *overload*를 사용했음을 살펴보도록 합니다.

2.17 기본값 인자

프로시저와 함수에 대한 인자에 기본값을 넣을 수 있습니다. 이 경우 이들 인자에 값을 넘기는 것을 무시할 수 있으며, 기본값을 사용할 것입니다.

다음 예제를 보도록 합니다.

```
program defaultparam;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

function SumAll(a, b: Integer; c: Integer = 0;
    d: Integer = 0): Integer;
begin
    result:= a + b + c + d;
end;
begin
    Writeln(SumAll(1, 2));
    Writeln(SumAll(1, 2, 3));
    Writeln(SumAll(1, 2, 3, 4));
    Write('Press enter key to close');
    Readln;
end.
```

이 예제에서 처음에는 두 개의 값을, 두번째는 세 개의 값을, 세번째는 네 개의 값을 사용하여 *SumAll* 함수를 세 번 호출했습니다. a와 b는 필수적인 인자이며, 기본값이 없기 때문에 모든 경우에 대해 값을 전달해야 합니다.

왼쪽부터 오른쪽으로 기본 인자 값의 전달을 무시할 수 있는데, 예를 들어 c에 대한 값 전달을 무시하려 한다면, d에 대한 값도 전달하면 안됩니다.

기본 인자는 오른쪽으로부터 왼쪽으로 시작해야 합니다. 다음 예제처럼 기본 인자 다음에 기본값이 없는 인자를 선언할 수는 없습니다.

```
function ErrorParameters(a: Integer; b: Integer = 10; c: Integer; x: string);
```

가장 중요한 인자는 함수나 프로시너의 머리 부분 왼쪽에 놓고, 덜 중요한 것(무시할 수 있는 것)은 함수나 프로시저의 오른쪽에 놓습니다.

2.18 정렬

데이터 정렬은 자료구조 주제의 일부분이며, 이 부분은 정렬을 구현하기 위한 프로시저와 함수를 필요로 하기 때문에 소개합니다.

정렬은 항상 배열과 리스트와 함께 사용합니다. 정수형을 저장하는 배열을 가지고 있고, 오름차순 혹은 내림차순으로 정렬하려 한다고 가정해봅니다. 이것을 각기 다른 방법을 이용해 처리할 수 있습니다.

2.19 버블 정렬 알고리즘

이 알고리즘은 단순한 정렬 알고리즘 중 하나입니다. 배열의 첫번째 요소를 두번째 요소와 비교해서 오름차순에서는 첫번째 요소가 두번째 요소보다 클 경우 서로 바꿉니다. 그 다음 두번째 요소와 세번째 요소를 비교하고, 이런식으로 배열의 마지막에 도달하기 전까지 계속 진행합니다. 그 후 배열이 이미 정렬되었다는 의미로 바꾸기 동작이 안 일어날 때까지 이 동작을 반복합니다.

배열에 다음 6개의 값을 지니고 있다고 가정해봅니다.

```
7
10
2
5
6
3
```

정렬을 하기 위해 한 바퀴 이상 돌아야 한다는 것을 알 것입니다.

첫번째 숫자를 두번째 것과 비교할 것이고 처음 것이 두번째 것보다 크면 바꾸기 동작이 일어날 것이며, 다섯번째 요소를 여섯번째 요소와 비교하기 전까지 두번째를 세번째와 비교할 것입니다.

처음 한 바퀴를 돌고 나면 배열은 다음과 같이 될 것입니다.

```
7
2
5
6
3
10
```

두번째 바퀴 다음

```
2
5
6
3
7
10
```

세번째

```
2
5
3
6
7
10
```

네번째

```
2
3
5
6
7
10
```

네 바퀴를 돌고 나서 정렬된 배열을 얻게 되는데, 네 바퀴 째에서는 바꾸기 동작이 일어나지 않기 때문입니다. 이는 이들 데이터를 정렬하는데 세 바퀴만 도는 것이 필요하지만, 네번째는 정렬의 발생을 확인하기 위해 사용한다는 것을 의미합니다.

작은 수가 큰 수 위로 거품처럼 떠다님을 알아 두도록 합니다.

다음은 버블 정렬 프로그램 입니다.

```
program BubbleSortProj;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes;

procedure BubbleSort(var X: array of Integer);
var
    Temp: Integer;
    i: Integer;
    Changed: Boolean;
begin
    repeat // Outer loop
        Changed:= False;
        for i:= 0 to High(X) - 1 do // Inner loop
            if X[i] > X[i + 1] then
            begin
                Temp:= X[i];
                X[i]:= X[i + 1];
                X[i + 1]:= Temp;
                Changed:= True;
            end;
    until not Changed;
end;

var
    Numbers: array [0 .. 9] of Integer;
    i: Integer;
begin
    Writeln('Please input 10 random numbers');
    for i:= 0 to High(Numbers) do
    begin
        Write('#', i + 1, ': ');
        Readln(Numbers[i]);
    end;
    BubbleSort(Numbers);
    Writeln;
    Writeln('Numbers after sort: ');
    for i:= 0 to High(Numbers) do
    begin
        Writeln(Numbers[i]);
    end;
```

```
    Write('Press enter key to close');
    Readln;
end.
```

보다 큰(>) 연산자를 보다 작은(<) 연산자로 바꾸어서 내림차순 정렬로 정렬 방식을 수정할 수 있습니다.

```
if X[i] > X[i + 1] then
```

다음 예제에서는 학생의 등급을 큰 점수부터 작은 점수순으로(내림 차순으로) 정렬하도록 하겠습니다.

2.20 학생 성적 정렬하기

```
program smSort; // Students' marks sort

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

type
    TStudent = record
        StudentName: string;
        Mark: Byte;
    end;

procedure BubbleSort(var X: array of TStudent);
var
    Temp: TStudent;
    i: Integer;
    Changed: Boolean;

begin
    repeat
        Changed:= False;
        for i:= 0 to High(X) - 1 do
            if X[i].Mark < X[i + 1].Mark then
```

```
                begin
                    Temp:= X[i];
                    X[i]:= X[i + 1];
                    X[i + 1]:= Temp;
                    Changed:= True;
                end;
        until not Changed;
end;
var
    Students: array [0 .. 9] of TStudent;
    i: Integer;

begin
    Writeln('Please input 10 student names and marks');
    for i:= 0 to High(Students) do
    begin
        Write('Student #', i + 1, ' name : ');
        Readln(Students[i].StudentName);

        Write('Student #', i + 1, ' mark : ');
        Readln(Students[i].Mark);
        Writeln;
    end;

    BubbleSort(Students);
    Writeln;
    Writeln('Marks after Bubble sort: ');
    Writeln('-----------------');

    for i:= 0 to High(Students) do
    begin
        Writeln('# ', i + 1, ' ', Students[i].StudentName,
            ' with mark (', Students[i].Mark, ')');
    end;
    Write('Press enter key to close');
    Readln;
end.
```

버블 정렬 알고리즘은 매우 단순하고, 프로그래머가 기억할 수 있습니다. 오직 데이터양이 적고 거의 정렬된 상태일 경우에만 적당합니다. 정렬되지 않은 많은 양의 데이터가 있다면 긴 시간이 걸리게 될 것이며, 다른 알고리즘을 사용하는 것이 더 좋습니다.

2.21 선택 정렬 알고리즘

이 유형의 정렬은 *버블 정렬*과 유사하지만, 많은 양의 데이터에 대해 더 빠릅니다. 외부 순환문과 내부 순환문 두 개의 순환문이 있습니다. 내부 순환문은 두 바퀴가 남기 전까지 외부 순환문을 돌 때마다 순환 횟수가 감소합니다.

```
program SelectionSort;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes
    { you can add units after this };

procedure SelectSort(var X: array of Integer);
var
    i: Integer;
    j: Integer;
    SmallPos: Integer;
    Smallest: Integer;
begin
    for i:= 0 to High(X) - 1 do // Outer loop
    begin
        SmallPos:= i;
        Smallest:= X[SmallPos];
        for j:= i + 1 to High(X) do // Inner loop
            if X[j] < Smallest then
        begin
            SmallPos:= j;
            Smallest:= X[SmallPos];
        end;
        X[SmallPos]:= X[i];
        X[i]:= Smallest;
    end;
end;

// Main application

var
    Numbers: array [0 .. 9] of Integer;
    i: Integer;

begin
    Writeln('Please input 10 random numbers');
```

```
    for i:= 0 to High(Numbers) do
    begin
        Write('#', i + 1, ': ');
        Readln(Numbers[i]);
    end;
    SelectSort(Numbers);
    Writeln;
    Writeln('Numbers after Selection sort: ');
    for i:= 0 to High(Numbers) do
    begin
        Writeln(Numbers[i]);
    end;
    Write('Press enter key to close');
    Readln;
end.
```

*버블 정렬*보다 빠름에도 불구하고, *선택 정렬*은 데이터 초기 정렬과 관계 없이 같은 순환 횟수를 지니는 반면에, *버블 정렬*은 데이터가 정렬되었거나 약간 정렬된 경우 빨라(외부 순환 횟수 줄어듦)집니다.

2.22 쉘 정렬 알고리즘

많은 양의 데이터가 있을 때 가장 빠른 정렬 방법이며, 데이터가 약간 정렬되어 있을 경우 *버블 정렬*과 유사하지만, 앞의 두 정렬 알고리즘보다는 좀 더 복잡합니다.

이 정렬 알고리즘의 이름은 창안자 도널드 쉘의 이름에서 따왔습니다.

```
program ShellSort;
{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes;

procedure ShellS(var X: array of Integer);
var
    Done: Boolean;
    Jump, j, i: Integer;
    Temp: Integer;

begin
    Jump:= High(X);
    while (Jump > 0) do // Outer loop
    begin
        Jump:= Jump div 2;
        repeat // Intermediate loop
            Done:= True;
            for j:= 0 to High(X) - Jump do // Inner loop
            begin
                i:= j + Jump;
                if X[j] > X[i] then // Swap
                begin
                    Temp:= X[i];
                    X[i]:= X[j];
                    X[j]:= Temp;
                    Done:= False;
                end;
            end; // end of inner loop
        until Done; // end of intermediate loop
    end; // end of outer loop
end;

var
    Numbers: array [0 .. 9] of Integer;
    i: Integer;

begin
    Writeln('Please input 10 random numbers');
```

```
    for i:= 0 to High(Numbers) do
    begin
        Write('#', i + 1, ': ');
        Readln(Numbers[i]);
    end;

    ShellS(Numbers);
    Writeln;
    Writeln('Numbers after Shell sort: ');

    for i:= 0 to High(Numbers) do
    begin
        Writeln(Numbers[i]);
    end;
    Write('Press enter key to close');
    Readln;
end.
```

2.23 문자열 정렬

정수를 정렬할 때 사용했던 것과 같은 비교 연산자 (> 와 <)를 사용하여 이름과 주소 등에 대해서도 정렬할 수 있습니다.

비교 동작은 왼쪽 문자부터 오른쪽 문자방향으로 일어나는데, 예를 들어 문자 *A*는 문자 *B*보다 작고 *'Ahmed'*라는 이름은 *'Badr'*보다 작습니다. 만약 두 개의 문자열의 첫번째 글자가 같다면, 그 다음 문자에 대해 비교를 진행합니다. *'Ali'* 와 *'Ahmed'*를 예를 들면, *h*는 *l*보다 작습니다.

2.24 학생 이름 정렬 프로그램

```
program smSort; // Students mark sort by name

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes;

type
    TStudent = record
        StudentName: string;
        Mark: Byte;
    end;

procedure BubbleSort(var X: array of TStudent);
var
    Temp: TStudent;
    i: Integer;
    Changed: Boolean;
begin
    repeat
        Changed:= False;
        for i:= 0 to High(X) - 1 do
            if X[i].StudentName > X[i + 1].StudentName then
            begin
                Temp:= X[i];
                X[i]:= X[i + 1];
                X[i + 1]:= Temp;
                Changed:= True;
            end;
    until not Changed;
end;
```

```
var
    Students: array [0 .. 9] of TStudent;
    i: Integer;
begin
    Writeln('Please input 10 student names and marks');
    for i:= 0 to High(Students) do
    begin
        Write('Student #', i + 1, ' name : ');
        Readln(Students[i].StudentName);
        Write('Student #', i + 1, ' mark : ');
        Readln(Students[i].Mark);
        Writeln;
    end;

    BubbleSort(Students);
    Writeln;
    Writeln('Marks after Bubble sort: ');
    Writeln('-----------------');

    for i:= 0 to High(Students) do
    begin
        Writeln('# ', i + 1, ' ', Students[i].StudentName,
            ' with mark (', Students[i].Mark, ')');
    end;
    Write('Press enter key to close');
    Readln;
end.
```

2.25 정렬 알고리즘 비교

이 예제에서는 큰 크기의 정수 배열을 가지고 이 장에서 언급한 세 가지 정렬 알고리즘을 비교할 것입니다. 각각의 알고리즘에 대해 걸린 시간을 측정하겠습니다.

```
program SortComparison;

{$mode objfpc}{$H+}
uses

    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Classes, SysUtils;

// Bubble sort

procedure BubbleSort(var X: array of Integer);
var
    Temp: Integer;
    i: Integer;
    Changed: Boolean;
begin
    repeat
        Changed:= False;
        for i:= 0 to High(X) - 1 do
            if X[i] > X[i + 1] then
            begin
                Temp:= X[i];
                X[i]:= X[i + 1];
                X[i + 1]:= Temp;
                Changed:= True;
            end;
    until not Changed;
end;

// Selection sort

procedure SelectionSort(var X: array of Integer);
var
    i: Integer;
    j: Integer;
    SmallPos: Integer;
    Smallest: Integer;
begin
    for i:= 0 to High(X) - 1 do // Outer loop
    begin
        SmallPos:= i;
        Smallest:= X[SmallPos];
        for j:= i + 1 to High(X) do // Inner loop
```

```
                if X[j] < Smallest then
                begin
                    SmallPos:= j;
                    Smallest:= X[SmallPos];
                end;
            X[SmallPos]:= X[i];
            X[i]:= Smallest;
        end;
end;

// Shell sort

procedure ShellSort(var X: array of Integer);
var
    Done: Boolean;
    Jump, j, i: Integer;
    Temp: Integer;
begin
    Jump:= High(X);
    while (Jump > 0) do // Outer loop
    begin
        Jump:= Jump div 2;
        repeat // Intermediate loop
            Done:= True;
            for j:= 0 to High(X) - Jump do // Inner loop
            begin
                i:= j + Jump;
                if X[j] > X[i] then // Swap
                begin
                    Temp:= X[i];
                    X[i]:= X[j];
                    X[j]:= Temp;
                    Done:= False;
                end;
            end; // inner loop
        until Done; // intermediate loop end
    end; // outer loop end
end;

// Randomize Data
procedure RandomizeData(var X: array of Integer);
var
    i: Integer;
begin
    Randomize;
    for i:= 0 to High(X) do
        X[i]:= Random(10000000);
end;
```

```
var
    Numbers: array [0 .. 59999] of Integer;
    i: Integer;
    StartTime: TDateTime;
begin
    Writeln('---------- Full random data');
    RandomizeData(Numbers);
    StartTime:= Now;
    Writeln('Sorting.. Bubble');
    BubbleSort(Numbers);
    Writeln('Bubble sort tooks (mm:ss): ',
        FormatDateTime('nn:ss', Now - StartTime));
    Writeln;

    RandomizeData(Numbers);
    Writeln('Sorting.. Selection');
    StartTime:= Now;
    SelectionSort(Numbers);
    Writeln('Selection sort tooks (mm:ss): ',
        FormatDateTime('nn:ss', Now - StartTime));
    Writeln;

    RandomizeData(Numbers);
    Writeln('Sorting.. Shell');
    StartTime:= Now;
    ShellSort(Numbers);
    Writeln('Shell sort tooks (mm:ss): ',
        FormatDateTime('nn:ss', Now - StartTime));
    Writeln;

    Writeln('---------- Nearly sorted data');
    Numbers[0]:= Random(10000);
    Numbers[High(Numbers)]:= Random(10000);
    StartTime:= Now;
    Writeln('Sorting.. Bubble');
    BubbleSort(Numbers);
    Writeln('Bubble sort tooks (mm:ss): ',
        FormatDateTime('nn:ss', Now - StartTime));
    Writeln;

    Numbers[0]:= Random(10000);
    Numbers[High(Numbers)]:= Random(10000);
    Writeln('Sorting.. Selection');
    StartTime:= Now;
    SelectionSort(Numbers);
    Writeln('Selection sort tooks (mm:ss): ',
        FormatDateTime('nn:ss', Now - StartTime));
    Writeln;
```

```
    Numbers[0]:= Random(10000);
    Numbers[High(Numbers)]:= Random(10000);
    Writeln('Sorting.. Shell');
    StartTime:= Now;
    ShellSort(Numbers);
    Writeln('Shell sort tooks (mm:ss): ',
        FormatDateTime('nn:ss', Now – Start,Time));

    Write('Press enter key to close');
    Readln;
end.
```

Chapter 3

그래픽 사용자 인터페이스

3.1 도입

그래픽 사용자 인터페이스(GUI)는 콘솔 인터페이스의 새로운 대안입니다. 이 인터페이스에는 *폼*, *버튼*, *메시지 상자*, *메뉴*, *체크 상자* 그리고 그래픽 구성요소들이 있습니다. 사용자들에게는 콘솔 프로그램보다는 GUI 프로그램이 사용하기가 더 쉽습니다.

GUI는 사업용 프로그램, 운영체제 프로그램, 게임, 라자루스와 같은 개발 도구, 그리고 기타 수많은 프로그램을 위해 쓰입니다.

3.2 우리의 첫번째 GUI 프로그램

새 GUI 프로그램을 만들려면 다음 라자루스 메뉴를 누릅니다.

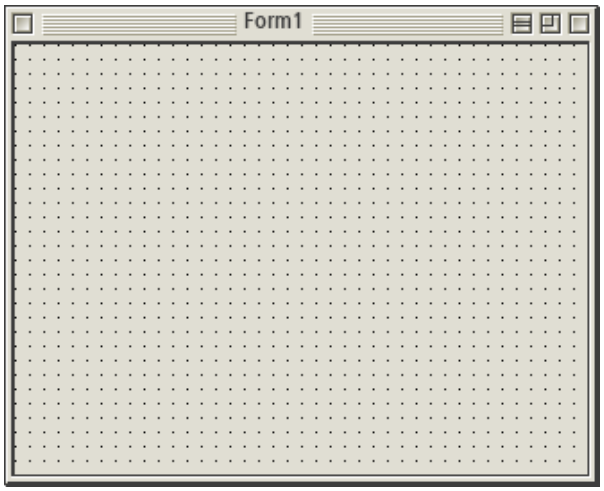
```
Project / New Project / Application
```

그 다음, 아래 메뉴를 따라가서 프로그램을 저장합니다.

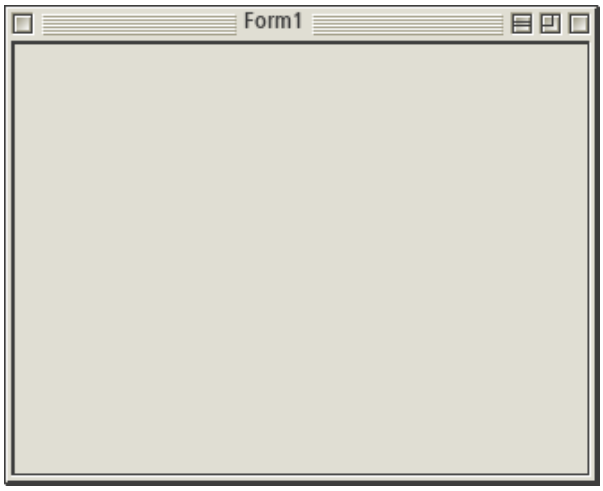
```
File / Save All
```

프로젝트 파일들을 저장하기 위해 *firstgui*와 같은 새로운 폴더를 만들 수 있습니다. 그 다음 *main.pas*와 같은 메인 Unit을 저장할 것이고, *firstgui.lpi*와 같은 프로젝트 이름을 정할 것입니다.

메인 Unit에서, 다음과 같은 메인 Unit과 관련된 폼을 보기 위해 *F12*키를 누를 수 있습니다.

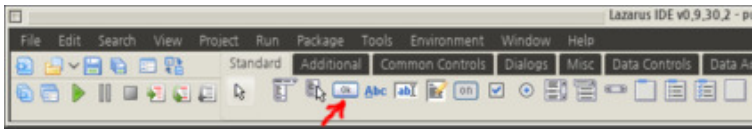


프로그램을 실행하면 다음 창을 보게 될 것입니다.

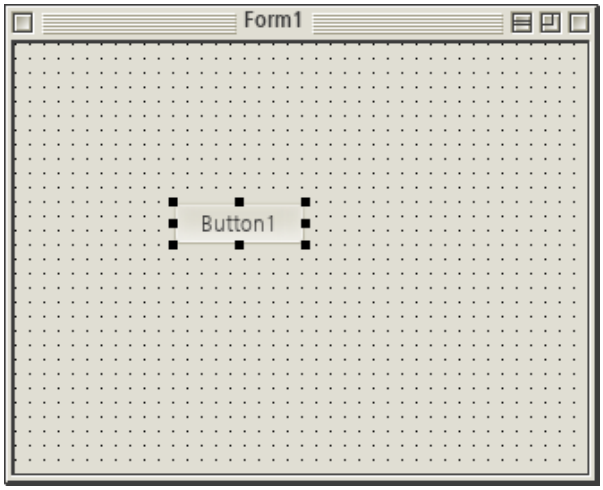


디자이너로 돌아가기 위해 프로그램을 끝낼 수 있습니다.

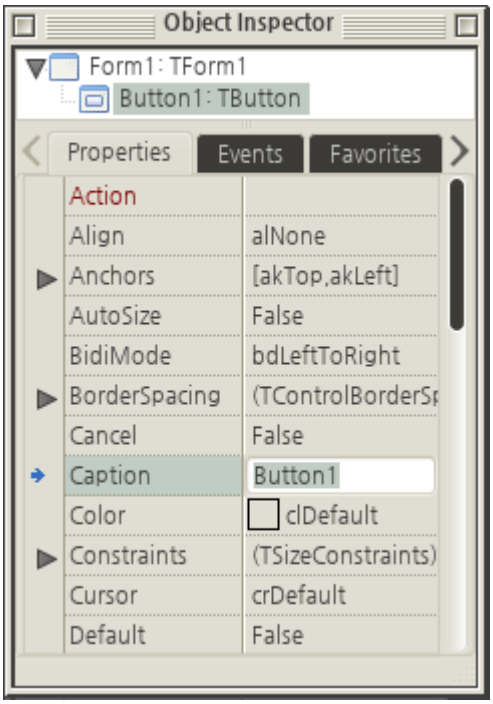
다음은 표준 구성요소 페이지로부터 폼 위에 *버튼*를 놓겠습니다.



그 다음 아래 그림과 같이 버튼를 임의의 위치에 놓아둡니다.



Object Inspector 창에서 버튼의 속성(캡션 이름, 폭 등)을 표시할 수 있습니다. 라자루스 창에서 나타나지 않는다면, 메인메뉴에서 *Window/Object Inspector* 를 마우스로 누르거나, *F11* 키를 눌러서 표시할 수 있습니다. 이제 다음 창을 보게 될 것입니다.



버튼의 속성을 수정하기 전에 폼이 아니라 *버튼*을 선택했는지 확인해야 합니다. 감독자 창에 첫번째에는 Properties, 두번째에는 Events 탭이 있는 것을 살펴보도록 합니다. 각각의 탭은 그 자신의 페이지를 보여줍니다.

이제 버튼에 표시된 텍스트를 *Caption* 속성을 눌러서 바꿀 수 있습니다.

그 다음, *Events* 페이지를 표시하기 위해 *Object Inspector*의 *Events* 탭을 선택하고, 메인 Unit에 코드 양식을 만들 OnClick 이벤트를 두 번 누릅니다.

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

이제 라자루스가 우리에게 만들어준 OnClick 이벤트 코드 양식에 다음 줄을 작성합니다.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage('Hello world, this is my first GUI application');
end;
```

이렇게 해서 프로그램을 실행하면, 마우스로 버튼를 눌렀을 때 기대하던 메시지가 들어있는 대화창을 표시할 것입니다.

리눅스에서는 *firstgui* 라는 실행 파일을 같은 디렉터리에서 찾을 것이고, 윈도우즈를 사용한다면 *firstgui.exe*를 같은 디렉터리에서 찾을 것입니다. 이들은 다른 컴퓨터로 복사할 수 있고, 라자루스 도구를 필요로 하지 않고 실행할 수 있는 파일들입니다.

앞의 예제에서 여러가지 중요한 점들이 있습니다.

1. 메인 프로그램 파일: 예제에서는 *firstgui.lpi* 파일로 저장했습니다. Project/Source를 마우스로 눌러서 볼 수 있습니다. 이 소스 코드를 보게 될 것입니다.

```
program firstgui;

{$mode objfpc}{$H+}

uses
    {$IFDEF UNIX}{$IFDEF UseCThreads}
    cthreads,
    {$ENDIF}{$ENDIF}
    Interfaces, // this includes the LCL widgetset
    Forms
    { you can add units after this }, main;
```

```
begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

특별한 목적으로 일부의 경우에 이 코드를 수정할 필요가 있겠지만, 대부분의 경우에는 라자루스가 자동으로 관리하도록 이대로 내버려둘 수 있습니다.

2. Main Unit: 프로그램을 실행할 때 자동으로 나타날 폼의 정의내용이 들어있는 Unit입니다.

아래는 *OnClick* 이벤트에 대해 완성한 코드의 main Unit입니다.

```
unit main;

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils, FileUtil, LResources, Forms, Controls,
    Graphics, Dialogs, StdCtrls;
type

    { TForm1 }

    TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    private
        { private declarations }
    public
        { public declarations }
    end;
var
    Form1: TForm1;
implementation

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage('Hello world, this is my first GUI application');
end;
```

```
initialization
    {$I main.lrs}
end.
```

메인 Unit 앞 부분에서 *레코드* 형식과 유사하지만 클래스인 *TForm1* 선언을 찾았습니다. 클래스에 대해서는 다음 장 "객체지향 프로그래밍"에서 이야기할 것입니다. *Button1*은 *TForm1* 클래스 안에 선언되어 있습니다.

이 Unit의 소스 코드는 *Ctrl+F12*를 누르고 *메인* Unit을 선택하면 볼 수 있습니다.

3. Object Inspector/Properties: Object Inspector의 이 페이지에서는 버튼의 캡션이나 위치, 폼의 색, 레이블의 글꼴 등과 같은 임의의 구성요소의 속성을 보고 수정할 수 있습니다. 이 구성요소들은 레코드와 유사하며, 이들의 속성은 레코드의 필드와 유사합니다.

4. Object Inspector/Events: Object Inspector의 이 페이지에서는 버튼의 *OnClick* 이벤트, 편집 상자에서의 *KeyPress*, 레이블의 *두번 누르기* 등과 같이 우리가 받을 수 있는 구성요소의 이벤트가 들어있습니다. 구성요소의 임의의 이벤트를 마우스로 누를 때, 라자루스는 이에 해당하는 이벤트가 발생할 때 호출될 코드를 작성하도록 우리를 위해 새로운 프로시저 양식 코드를 만듭니다. 버튼의 *OnClick* 이벤트에 대한 아래 예제처럼 해당하는 이벤트에 대한 파스칼 코드를 작성할 수 있습니다.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage('Hello world, this is my first GUI application');
end;
```

이 프로시저는 사용자가 이 버튼를 눌렀을 때 호출됩니다. 이 프로시저를 이벤트 핸들러라고 부릅니다.

3.3 두번째 GUI 프로그램

이 예제에서는 사용자에게 편집 상자로 이름을 입력하게 하고, 버튼를 누르면 레이블에 인사 메시지를 표시하도록 할 것입니다.

이 프로그램을 작성하기 위해 다음을 따릅니다.

- 새 프로그램을 만들고 *inputform*으로 저장합니다. 메인 Unit은 *main.pas*로 저장하고, 표준 구성요소 탭으로부터 이 구성요소를 놓아둡니다.

2개의 레이블
편집 상자
버튼

이제 위 구성요소들의 속성을 다음과 같이 바꿉니다.

Form1:
Name: fmMain
Caption: Input from application

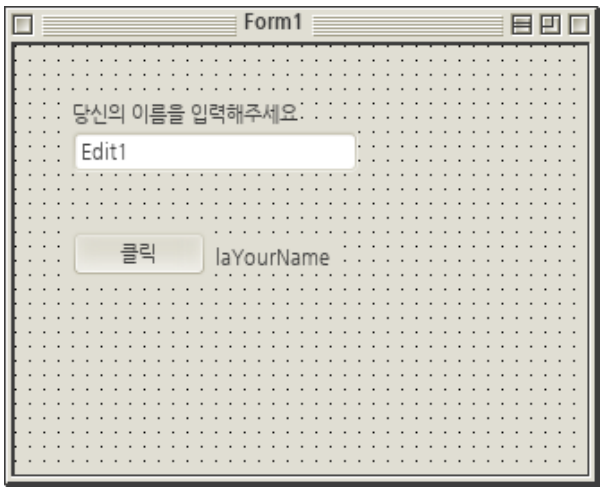
Label1:
Caption: 당신의 이름을 입력해주세요

Label2:
Name:laYourName

Edit1:
Name: edName
Text:

Button1:
Caption: 클릭

아래 그림과 같이 폼 위에 구성 요소를 놓아둡니다.



이 코드를 *OnClick* 이벤트 핸들러에 넣습니다.

```
procedure TfmMain.Button1Click(Sender: TObject);
begin
    laYourName.Caption:= 'Hello ' + edName.Text;
end;
```

앞의 예제에서, 사용자가 입력한 내용을 읽기 위해 편집 상자 *edName*에 있는 *Text* 필드를 사용했습니다. 이는 사용자의 입력을 받기 위해 콘솔 프로그램에서 사용한 *Readln* 프로시저에 대한 그래픽 환경의 대안책입니다.

또한, 메시지를 표시하기 위해 레이블 *laYourName*에 있는 *Caption* 속성을 사용했습니다. 이것은 GUI 프로그램에서 *Writeln* 대신 사용하는 방법 중 하나입니다.

3.4 ListBox 프로그램

다음 예제에서는 목록 상자에 텍스트를 추가하고, 삭제하며, 목록을 지우려고 합니다. 이 프로그램을 만들기 위해 다음 순서를 따릅니다.

- 새 프로그램을 만들고 메인 폼에 네 개의 버튼과 편집 상자, 목록 상자(*TListBox*)를 놓아둡니다.

- 버튼들의 이름을 아래 이름들로 바꿉니다.

```
btAdd, btClear, btDelete, btClose
```

- 아래 그림대로 버튼의 이름을 따라 캡션을 바꿉니다.

- 버튼의 *OnClick* 이벤트에 대해 이들 이벤트 핸들러를 작성합니다.

```
procedure TForm1.btAddClick(Sender: TObject);
begin
    ListBox1.Items.Add(Edit1.Text);
end;

procedure TForm1.btClearClick(Sender: TObject);
begin
    ListBox1.Clear;
end;
```

```
procedure TForm1.btDeleteClick(Sender: TObject);
var
    Index: Integer;
begin
    Index:= ListBox1.ItemIndex;
    if Index <> -1 then
        ListBox1.Items.Delete(Index);
end;

procedure TForm1.btCloseClick(Sender: TObject);
begin
    Close;
end;
```

추가 버튼를 누르면 편집 상자의 텍스트가 목록에 삽입될 것입니다. 삭제 버튼를 누르면 현재 선택한 항목을 삭제할 것입니다. 초기화 버튼를 누르면 모든 목록을 지울 것입니다. 마지막으로 닫기 버튼를 누르면 프로그램을 닫을 것입니다.

3.5 텍스트 편집기 프로그램

이 예제에서는 간단한 텍스트 편집기를 어떻게 만드는지 보여드리겠습니다.

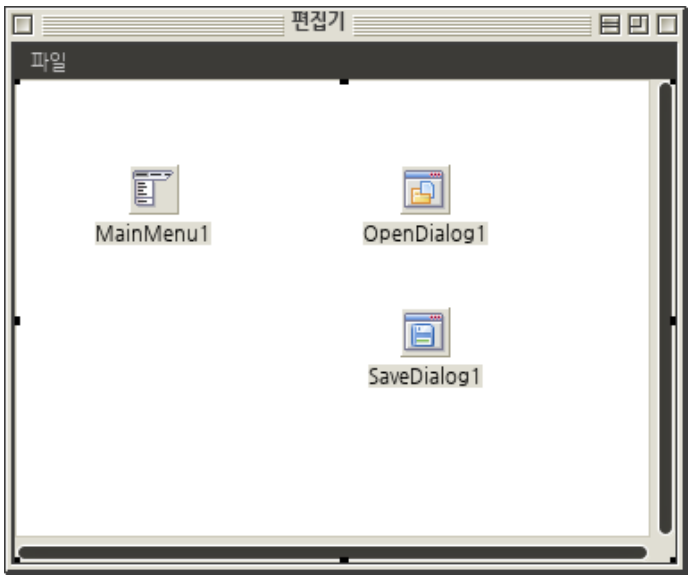
다음 순서를 따릅니다.

- 새 프로그램을 만들고 메인 폼에 아래 구성 요소를 넣습니다.

  1. TMainMenu
  2. TMemo: 정렬 속성을 *alClient*로 바꾸고, 스크롤 바 속성을 *ssBoth*로 바꿉니다.
  3. 구성요소 팔레트의 대화상자 페이지에 있는 TOpenDialog와 TSaveDialog

- *MainMenu1* 구성요소를 마우스로 두 번 누르고 *File* 메뉴와 *Open File*, *Save File*, *Close* 하위 메뉴 항목을 추가합니다.

아래와 같은 폼을 보게 될 것입니다.



- *Open File* 항목의 *OnClick* 이벤트에 대해 이 코드를 작성합니다.

```
if OpenDialog1.Execute then
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

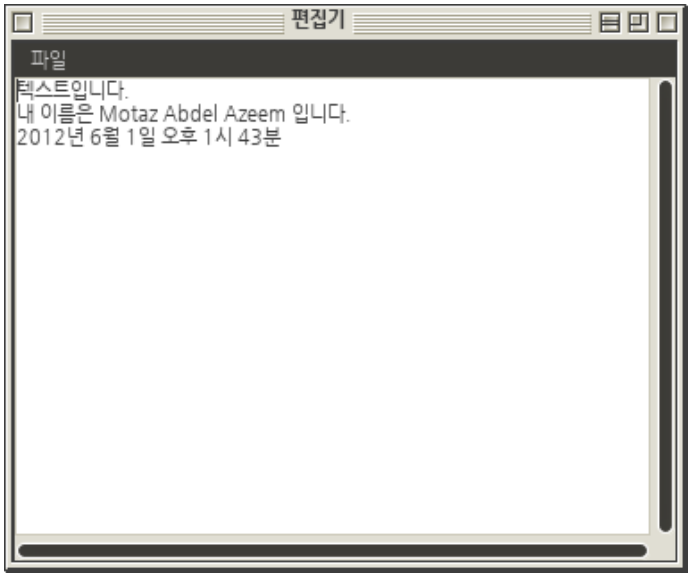
- *Save File* 항목에 대해 이 코드를 작성합니다.

```
if SaveDialog1.Execute then
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

- *Close* 항목에 대해 이 코드를 작성합니다.

```
Close;
```

이 텍스트 편집기 프로그램을 실행한 다음 텍스트를 작성하고 디스크에 저장할 수 있습니다. 또한 *.pas* 파일과 같은 기존의 텍스트 파일을 열 수 있습니다.



3.6 뉴스 프로그램

이제 다음 순서에 따라 뉴스 제목을 저장하기 위한 프로그램을 작성하고자 합니다.

- 새 프로그램을 만들고 *gnews*라고 이름 짓습니다.
- *TButton* 형의 두 개의 버튼를 추가합니다.
- 텍스트 상자(*TEdit*)를 추가합니다.
- 메모(*TMemo*)를 추가합니다.
- 다음 값들을 따라 구성 요소 값을 바꿉니다.

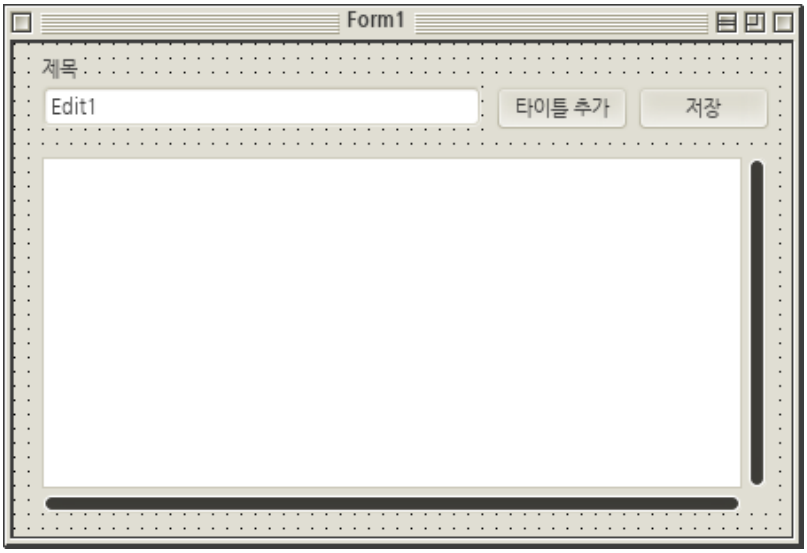
Button1:
Caption: 타이틀 추가

Button2:
Caption: 저장
Anchors: Left=False, Right=True

Edit1:
Text:

Memo1:
ScrollBars: ssBoth
ReadOnly: True
Anchors: Top=True, Left=True, Right=True, Bottom=True

이제 아래와 같은 폼을 보게 될 것입니다.



- *Add Title* 버튼의 *OnClick* 이벤트에 대해 이 코드를 작성합니다.

```
Memo1.Lines.Insert(0,
    FormatDateTime('yyyy-mm-dd hh:nn', Now) + ':' + Edit1.Text);
```

- *Save* 버튼에 대해 이 코드를 작성합니다.

```
Memo1.Lines.SaveToFile('news.txt');
```

- 입력한 뉴스를 저장하기 위해 메인 폼의 *OnClose* 이벤트에 대해 이 코드를 작성합니다.

```
Memo1.Lines.SaveToFile('news.txt');
```

- 메인 폼의 *OnCreate* 이벤트에 대해, 내용이 존재하는 경우 뉴스 제목을 저장하기 전에 먼저 불러올 수 있는 코드를 작성합니다.

```
if FileExists('news.txt') then
    Memo1.Lines.LoadFromFile('news.txt');
```

3.7 두번째 폼을 가진 프로그램

앞의 GUI 프로그램에서는 한 개의 폼만 사용했지만, 때로는 실제 프로그램에서 여러 개의 폼이 필요하기도 합니다.

여러 개의 폼을 가진 GUI 프로그램을 작성하려면, 다음 순서를 따릅니다.

1. 새 프로그램을 만들고, *secondform*이라는 새로운 폴더에 저장합니다.
2. 메인 Unit을 *main.pas*로 저장하고, 폼 구성요소의 이름을 *fmMain*으로 합니다. *secondform.lpi*로 저장합니다.
3. *File / New Form*을 눌러서 새 폼을 추가합니다. 새 Unit을 *second.pas*로 저장하고 폼 구성요소의 이름을 *fmSecond*로 합니다.
4. 두번째 폼에 레이블을 추가하고, *Caption* 속성에 **'Second Form'**으로 적습니다. 레이블의 *Font.size* 속성에서 글꼴 크기를 늘입니다.
5. 메인 폼으로 돌아가서 버튼을 놓아둡니다.
6. 메인 Unit의 *implementation* 섹션 다음에 다음 줄을 추가합니다.

```
uses second;
```

7. 버튼의 *OnClick* 이벤트에 대해 이 코드를 작성합니다.

```
    fmSecond.Show;
```

이제 프로그램을 실행하고 버튼을 눌러 두번째 폼을 표시해보도록 합니다.

Chapter 4

객체지향 프로그래밍

4.1 도입

객체지향 프로그래밍에서는 프로그램 전체를 객체로 설명합니다. 예를 들어, 자동차 정보는 모델 이름, 모델 연도, 가격이 들어있는 객체로 표현하며, 이 객체는 파일에 데이터를 저장할 수 있는 능력이 있습니다.

객체에는 다음 요소가 있습니다.

1. 상태 정보를 저장하는 **속성**입니다. 이 값들은 변수에 저장합니다.
2. **메서드**라고 부르는 프로시저와 함수입니다. 이 메서드들은 이 객체가 할 수 있는 동작을 나타냅니다.
3. **이벤트**: 객체위로 마우스를 가져간다든지, 마우스 버튼를 누른다든지 등의 이 이벤트들은 객체가 받을 수 있습니다.
4. **이벤트 핸들러**: 이벤트가 발생했을 때 실행하는 프로시저들입니다.

서로 관련된 속성과 메서드들은 객체로 표현할 수 있습니다.

객체 = 코드 + 데이터

객체지향 프로그래밍의 예로 앞 장에서 우리가 사용했던 GUI가 있습니다. 이 장에서는 *버튼*, *레이블*, 폼과 같은 많은 객체들을 사용할 것입니다. 각각의 객체에는 *Caption*, *Width*와 같은 속성이 있고, *Show*, *Hide*, *Close* 등과 같은 메서드들이 있습니다. 또한 *OnClick*, *OnCreate*, *OnClose* 등과 같은 이벤트도 지니고 있습니다. 코드는 이벤트 핸들러를 나타내는 *OnClick*과 같은 특정 이벤트에 반응하기 위해 프로그래머가 작성합니다.

4.2 첫번째 예제: 날짜와 시간

날짜와 시간에 대해 수행하는 동작과 날짜 시간이 들어있는 객체를 작성하도록 하겠습니다. *DateTimeUnit*이라고 하는 새 Unit을 만들고, *TmyDateTime*이라고 하는 클래스를 작성하도록 하겠습니다.

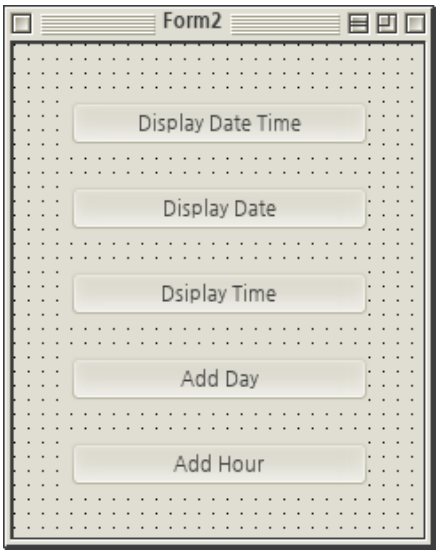
*클래스*는 객체 형식입니다. 클래스를 사용하려면, 클래스의 인스턴스를 선언해야 합니다. 이 인스턴스를 *객체*라고 하며, *Integer*와 *string* 형으로 다루던 것과 동일합니다. 클래스의 인스턴스를 사용하려면, 이들을 변수(*I*, *J*, *Address* 등)로 선언합니다.

다음은 Unit 코드입니다.

```
unit DateTimeUnit;

{$mode objfpc}{$H+}

interface
uses
    Classes, SysUtils;

type

    { TMyDateTime }

    TMyDateTime = class
    private
        fDateTime: TDateTime;
    public
        function GetDateTime: TDateTime;
        procedure SetDateTime(ADateTime: TDateTime);
        procedure AddDays(Days: Integer);
        procedure AddHours(Hours: Single);
        function GetDateTimeAsString: string;
        function GetTimeAsString: string;
        function GetDateAsString: string;
        constructor Create(ADateTime: TDateTime);
        destructor Destroy; override;
    end;

implementation

{ TMyDateTime }

function TMyDateTime.GetDateTime: TDateTime;
begin
    Result:= fDateTime;
end;

procedure TMyDateTime.SetDateTime(ADateTime: TDateTime);
begin
    fDateTime:= ADateTime;
end;

procedure TMyDateTime.AddDays(Days: Integer);
begin
    fDateTime:= fDateTime + Days;
end;
```

```
procedure TMyDateTime.AddHours(Hours: Single);
begin
    fDateTime:= fDateTime + Hours / 24; end;

function TMyDateTime.GetDateTimeAsString: string;
begin
    Result:= DateTimeToStr(fDateTime); end;

function TMyDateTime.GetTimeAsString: string;
begin
    Result:= TimeToStr(fDateTime); end;

function TMyDateTime.GetDateAsString: string;
begin
    Result:= DateToStr(fDateTime); end;

constructor TMyDateTime.Create(ADateTime: TDateTime);
begin
    fDateTime:= ADateTime;
end;

destructor TMyDateTime.Destroy;
begin
    inherited Destroy;
end;
end.
```

다음과 같이 메인 폼에 다섯 개의 버튼을 놓습니다.



각각의 단추의 *OnClick* 이벤트에 다음 코드를 작성하도록 합니다.

```
unit main;

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils, FileUtil, LResources, Forms, Controls,
    Graphics, Dialogs, StdCtrls, DateTimeUnit;

type
    { TForm1 }

    TForm1 = class(TForm)
        Button1: TButton;
        Button2: TButton;
        Button3: TButton;
        Button4: TButton;
        Button5: TButton;
        procedure Button1Click(Sender: TObject);
        procedure Button2Click(Sender: TObject);
        procedure Button3Click(Sender: TObject);
        procedure Button4Click(Sender: TObject);
        procedure Button5Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        private
            { private declarations }
        public
            MyDT: TMyDateTime;
            { public declarations }
    end;
var
    Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.FormCreate(Sender: TObject);
begin
    MyDT:= TMyDateTime.Create(Now);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage(MyDT.GetDateTimeAsString);
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    ShowMessage(MyDT.GetDateAsString);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    ShowMessage(MyDT.GetTimeAsString);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
    MyDT.AddHours(1);
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
    MyDT.AddDays(1);
end;

initialization
    {$I main.lrs}

end.
```

이 예제에서 다음의 중요한 사실들을 살펴보겠습니다.

DateTimeUnit Unit에서는

1. *TmyDateTime*을 새로운 클래스를 정의하는 키워드인 *class*로 정의했습니다.
2. *Create*라고 하는 *constructor* 메서드를 도입했습니다. 메모리에 객체를 만들고 초기화 할 때 사용하는 특별한 프로시저 입니다.
3. *Destroy*라고 하는 *destructor* 메서드를 도입했습니다. 객체를 사용을 마무리 짓고 나서 객체의 메모리에서 없앨 때 사용하는 특별한 프로시저 입니다.
4. 클래스에 두 가지 섹션이 있습니다.
   private: 클래스 Unit 밖에서 접근할 수 없는 속성이나 메서드를 포함합니다. 다른 섹션은 *public* 입니다: Unit 밖에서 접근할 수 있는 속성이나 메서드를 포함합니다. 클래스에 public 섹션이 없다면 모든 곳에서 사용할 수 없음을 의미합니다.

메인 Unit 에서는

1. *DateTimeUnit*에 접근하기 위해 메인 폼의 *uses* 절에 이 클래스를 추가했습니다.

2. Unit 안에 *MyDT* 객체를 선언했습니다.

```
MyDT: TmyDateTime;
```

3. 메인 폼의 *OnCreate* 이벤트에 객체를 만들고 초기화 했습니다.

```
MyDT:= TmyDateTime.Create(Now);
```

이는 오브젝트 파스칼 언어에서 객체를 만드는 메서드입니다.

4.3 객체지향 파스칼의 뉴스 프로그램

이 예제에서는 객체지향 기법을 사용하여 뉴스 프로그램을 재작성하고자 합니다. 또한 뉴스를 여러 개의 파일로 분류해야 합니다.

새로운 GUI 프로그램을 만들었고 *oonews*라고 이름 지었습니다.

다음 예제는 뉴스 기능이 있는 *TNews* 클래스를 포함한 새 Unit입니다.

```
unit news;

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils;

type
    TNewsRec = record
        ATime: TDateTime;
        Title: string[100];
    end;

    { TNews }

    TNews = class
    private
        F: file of TNewsRec;
        fFileName: string;
    public
        constructor Create(FileName: string);
        destructor Destroy; override;
        procedure Add(ATitle: string);
        procedure ReadAll(var NewsList: TStringList);
        function Find(Keyword: string;
            var ResultList: TStringList): Boolean;
    end;

implementation

{ TNews }

constructor TNews.Create(FileName: string);
begin
    fFileName:= FileName;
end;
```

```
destructor TNews.Destroy;
begin
    inherited Destroy;
end;

procedure TNews.Add(ATitle: string);
var
    Rec: TnewsRec; begin
    AssignFile(F, fFileName);
    if FileExists(fFileName) then
    begin
        FileMode:= 2; // Read/write access
        Reset(F);
        Seek(F, FileSize(F));
    end

    else
        Rewrite(F);

    Rec.ATime:= Now;
    Rec.Title:= ATitle;
    Write(F, Rec);
    CloseFile(F);

end;

procedure TNews.ReadAll(var NewsList: TStringList);
var
    Rec: TnewsRec;
begin
    NewsList.Clear;
    AssignFile(F, fFileName);
    if FileExists(fFileName) then
    begin
        Reset(F);
        while not Eof(F) do
        begin
            Read(F, Rec);
            NewsList.Add(DateTimeToStr(Rec.ATime) + ' : ' + Rec.Title);
        end;
        CloseFile(F);
    end;

end;
```

```
function TNews.Find(Keyword: string; var ResultList: TStringList): Boolean;
var
    Rec: TnewsRec;
begin
    ResultList.Clear;
    Result:= False;
    AssignFile(F, fFileName);
    if FileExists(fFileName) then
    begin
        Reset(F);
        while not Eof(F) do
        begin
            Read(F, Rec);
            if Pos(LowerCase(Keyword), LowerCase(Rec.Title)) > 0 then
            begin
            ResultList.Add(DateTimeToStr(Rec.ATime) + ' : ' + Rec.Title);
            Result:= True;
            end;
        end;
        CloseFile(F);
    end;
end;

end.
```

메인 폼에 다음과 같이 *편집 상자*, *콤보 상자*, 세 개의 *버튼*, *메모*, 두 개의 *레이블* 구성요소를 추가했습니다.

메인 Unit에는 이 코드를 작성했습니다.

```
unit main;

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
    Dialogs, News, StdCtrls;

type

    { TForm1 }

    TForm1 = class(TForm)
        btAdd: TButton;
        btShowAll: TButton;
        btSearch: TButton;
        cbType: TComboBox;
        edTitle: TEdit;
        Label1: TLabel;
        Label2: TLabel;
        Memo1: TMemo;
        procedure btAddClick(Sender: TObject);
        procedure btSearchClick(Sender: TObject);
        procedure btShowAllClick(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { private declarations }
    public
        NewsObj: array of TNews;
        { public declarations }
    end;

var
    Form1: TForm1;

implementation

{ TForm1 }

procedure TForm1.FormCreate(Sender: TObject);
var
    i: Integer;
begin
```

```
    SetLength(NewsObj, cbType.Items.Count);
    for i:= 0 to High(NewsObj) do
        NewsObj[i]:= TNews.Create(cbType.Items[i] + '.news');
end;

procedure TForm1.btAddClick(Sender: TObject);
begin
    NewsObj[cbType.ItemIndex].Add(edTitle.Text);
end;

procedure TForm1.btSearchClick(Sender: TObject);
var
    SearchStr: string;
    ResultList: TstringList;
begin
    ResultList:= TStringList.Create;
    if InputQuery('Search News', 'Please input keyword', SearchStr) then
        if NewsObj[cbType.ItemIndex].Find(SearchStr, ResultList) then
        begin
            Memo1.Lines.Clear;
            Memo1.Lines.Add(cbType.Text + ' News');
            Memo1.Lines.Add('----------');
            Memo1.Lines.Add(ResultList.Text);
        end
    else
        Memo1.Lines.Text:= SearchStr + ' not found in ' +
            cbType.Text + ' news';
    ResultList.Free;
end;

procedure TForm1.btShowAllClick(Sender: TObject);
var
    List: TStringList;
begin
    List:= TStringList.Create;
    NewsObj[cbType.ItemIndex].ReadAll(List);
    Memo1.Lines.Clear;
    Memo1.Lines.Add(cbType.Text + ' News');
    Memo1.Lines.Add('----------');
    Memo1.Lines.Add(List.Text);
    List.Free;
end;

procedure TForm1.FormClose(Sender: TObject;
    var CloseAction: TCloseAction);
var
    i: Integer;
```

```
begin
    for i:= 0 to High(NewsObj) do
        NewsObj[i].Free;

    NewsObj:= nil;

end;

initialization
    {$I main.lrs}

end.
```

앞의 예제에서 다음 사실들을 살펴보겠습니다.

1. 사용 여부에 따라 할당하고, 확장하고, 축소하고, 없앨 수 있는 동적 배열을 사용했습니다. news 객체의 동적 배열을 다음과 같이 선언했습니다.

    ```
    NewsObj: array of TNews;
    ```

    실행 시간동안 사용하기 전에 *SetLength* 프로시저를 사용하여 초기화해야 합니다.

    ```
    SetLength(NewsObj, 10);
    ```

    이는 배열에 10개 요소를 할당한다는 의미입니다. 이는 일반 배열의 선언과 유사합니다.

    ```
    NewsObj: array [0 .. 9] of TNews;
    ```

    일반적인 배열의 크기는 프로그램이 실행하는 동안 항상 정해진 그대로 남아 있지만, 동적 배열의 크기는 늘였다 줄였다 할 수 있습니다.

    이 예제에서는 콤보 상자에 존재하는 분류에 따라 배열을 초기화합니다.

    ```
    SetLength(NewsObj, cbType.Items.Count);
    ```

    *ComboBox.Items*에 더 많은 분류를 추가하면, 동적 배열의 크기는 이에 따라 증가할 것입니다.

2. *TNews* 형은 클래스이며, 이것을 바로 사용할 수 없습니다. *NewsObj*와 같이 클래스의 객체 인스턴스를 선언해야 합니다.

3. 프로그램의 마지막에는 객체를 릴리스하고, 그 다음 동적 배열을 릴리스했습니다.

```
for i:= 0 to High(NewsObj) do
    NewsObj[i].Free;

NewsObj:= nil;
```

4.4 큐 프로그램

큐는 자료구조 형의 한 예입니다. 요소를 순서대로 삽입하고 저장하기 위해 사용하며, 삽입한 순서대로 삭제합니다. 이러한 규칙을 *선입선출(First-In-First-Out)* 이라고 합니다.

이 예제에서는 *TQueue* 클래스가 있는 *Queue*라고 하는 Unit을 작성합니다. *TQueue* 클래스는 이름과 같은 데이터를 저장하는데 사용할 수 있으며, 순서대로 데이터를 가져올 수 있습니다. 큐로부터 데이터를 가져오면 큐에서 데이터를 삭제합니다. 예를 들어 큐에 10개의 항목이 들어있을때, 3개의 항목을 가져오면, 큐에는 7개의 항목이 남아있을 것입니다.

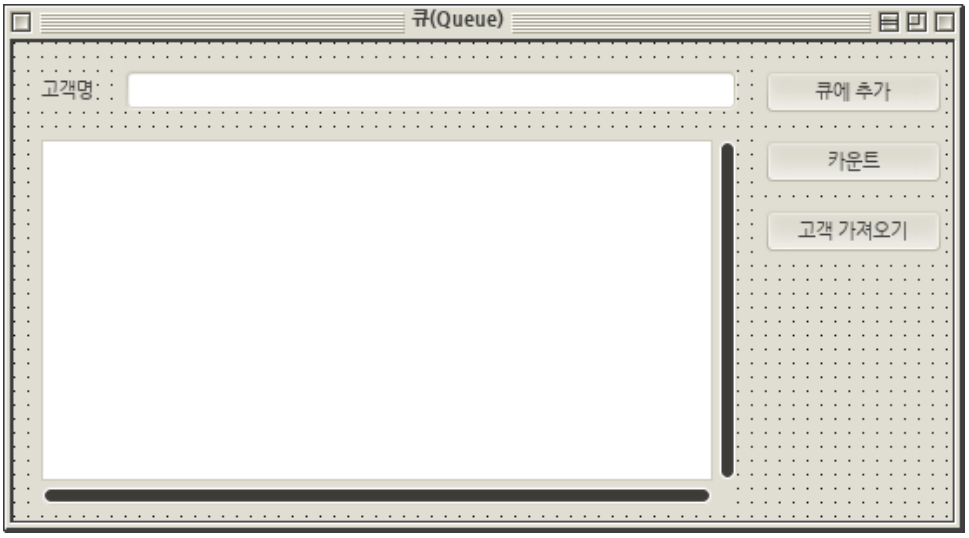
Queue Unit 입니다.

```
unit queue;
// This unit contains TQueue class,
// which is suitable for any string queue

{$mode objfpc}{$H+}

interface
uses
    Classes, SysUtils;

type

    { TQueue }

    TQueue = class
    private
        fArray: array of string;
        fTop: Integer;
    public
        constructor Create;
        destructor Destroy; override;
        function Put(AValue: string): Integer;
        function Get(var AValue: string): Boolean;
        function Count: Integer;
        function ReOrganize: Boolean;
    end;

implementation

{ TQueue }

constructor Tqueue.create;
begin
    fTop:= 0;
end;
```

```
destructor TQueue.destroy;
begin
    SetLength(fArray, 0); // Erase queue array from memory
    inherited destroy;
end;

function TQueue.Put(AValue: string): Integer; begin
    if fTop >= 100 then
        ReOrganize;

    SetLength(fArray, Length(fArray) + 1);
    fArray[High(fArray)]:= AValue;
    Result:= High(fArray) - fTop;
end;

function TQueue.Get(var AValue: string): Boolean;
begin
    AValue:= '';
    if fTop <= High(fArray) then
    begin
        AValue:= fArray[fTop];
        Inc(fTop);
        Result:= True;
    end
    else // empty
    begin
        Result:= False;

        // Erase array
        SetLength(fArray, 0);
        fTop:= 0;
    end;
end;

function TQueue.Count: Integer;
begin
    Result:= Length(fArray) - fTop;
end;

function TQueue.ReOrganize: Boolean;
var
    i: Integer;
    PCount: Integer;
begin
    if fTop > 0 then
    begin
```

```
            PCount:= Count;
            for i:= fTop to High(fArray) do
                fArray[i - fTop]:= fArray[i];
            // Truncate unused data
            setLength(fArray, PCount);
            fTop:= 0;
            Result:= True; // Re Organize is done
        end
        else
            Result:= False; // nothing done
        end;
end;

end.
```

큐 프로그램의 메인 폼입니다.



메인 Unit의 코드입니다.

```
unit main;

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
    Dialogs, Queue, StdCtrls;
```

```
type

    { TfmMain }

    TfmMain = class(TForm)
        bbAdd: TButton;
        bbCount: TButton;
        bbGet: TButton;
        edCustomer: TEdit;
        Label1: TLabel;
        Memo1: TMemo;
        procedure bbAddClick(Sender: TObject);
        procedure bbCountClick(Sender: TObject);
        procedure bbGetClick(Sender: TObject);
        procedure FormClose(Sender: TObject;
            var CloseAction: TCloseAction);
        procedure FormCreate(Sender: TObject);
    private
        { private declarations }
    public
        MyQueue: TQueue;
        { public declarations }
    end;

var
    fmMain: TfmMain;

implementation

{ TfmMain }

procedure TfmMain.FormCreate(Sender: TObject);
begin
    MyQueue:= TQueue.Create;
end;

procedure TfmMain.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
    MyQueue.Free;
end;

procedure TfmMain.bbCountClick(Sender: TObject);
begin
    Memo1.Lines.Add('Queue length is: ' + IntToStr(MyQueue.Count));
end;
```

```
procedure TfmMain.bbAddClick(Sender: TObject);
var
    APosition: Integer;
begin
    APosition:= MyQueue.Put(edCustomer.Text);
    Memo1.Lines.Add(edCustomer.Text + ' has been added as # ' +
        IntToStr(APosition + 1));
end;

procedure TfmMain.bbGetClick(Sender: TObject);
var
    ACustomer: string;
begin
    if MyQueue.Get(ACustomer) then
    begin
        Memo1.Lines.Add('Got: ' + ACustomer + ' from the queue');
    end
    else
        Memo1.Lines.Add('Queue is empty');
end;

initialization
    {$I main.lrs}

end.
```

TQueue 클래스에서 동적 배열을 확장하여 새로운 항목을 추가하기 위해 *Put* 메서드를 사용했고, 배열의 마지막 요소에 새로운 항목을 삽입했습니다.

큐의 위 꼭대기에서 항목을 제거하기 위해 *Get* 메서드를 호출할 때, *fTop* 포인터는 큐의 다음 항목으로 이동할 것입니다.

동적 배열의 꼭대기에서 항목을 제거하는 동작은 *fTop* 포인터가 이동하는 결과를 가져다주지만, 동적 배열의 항목은 메모리에 그대로 남아 있을 것이고, 공간을 차지할 것입니다. 왜냐하면, 동적 배열의 바닥부터만 지울 수 있기 때문에, 항목 갯수가 100개에 도달하면 동적 배열의 꼭대기에 있는 큐 요소를 이동하기 위해 *ReOrganize* 메서드를 호출할 것이며, 이렇게 하여 배열의 사용하지 않는 요소를 삭제하기 때문입니다.

동적 배열의 꼭대기 방향으로 큐의 요소들을 이동하는 코드입니다.

```
    for i:= fTop to High(fArray) do
        fArray[i - fTop]:= fArray[i];
```

그리고 바닥으로부터 동적 배열을 자르는 코드입니다.

```
// Truncate unused data
setLength(fArray, PCount);
fTop:= 0;
```

이 예제에서 정보 은닉을 도입한 객체지향 프로그래밍을 알았습니다. 변수로의 접근과 같은 민감한 데이터로의 접근을 거부할 것입니다. 대신 객체에 대한 이상한 동작을 하지 않을 특정 메서드를 사용할 수 있습니다.

민감한 데이터나 메서드는 클래스 선언의 private 섹션에 위치할 것입니다.

```
private
    fArray: array of string;
    fTop: Integer;
```

이 클래스를 사용하는 프로그래머는 이 변수들에 직접 접근할 수 없습니다. 만약 이 변수에 접근할 수 있었다면, 고의적으로 *fTop* 이나 *fArray* 를 수정하여 큐를 깨뜨릴 수 있습니다. 예를 들어 큐에 고작 10개의 요소를 가지고 있었는데 *fTop*을 1000으로 수정했다면, 실행 도중에 접근 위반(Access violation) 오류가 발생할 것입니다.

직접 변수 사용의 대안으로서, 배열로부터 안전하게 항목을 추가하고 제거하기 위한 메서드 *Put*과 *Get*을 구현했습니다. 이 메서드는 요소들 내부를 제어하기 위한 게이트를 사용하는 것과 같습니다. OOP의 이러한 특징을 *캡슐화(encapsulation)* 라고 합니다.

4.5 객체지향 파일

첫번째 장에서 제각기 다른 형식의 파일을 사용했고, 구조적 프로그래밍(프로시저와 함수)을 사용하여 이 파일들을 다루었습니다. 이제 파일 객체를 사용하여 파일에 접근할 때입니다.

오브젝트 파스칼의 파일 클래스 중 하나는 파일들을 다루기 위한 메서드와 속성을 가진 *TFileStream* 입니다.

이 방법을 사용하면 좀 더 표준적이고 프로그래머들이 예측하기 쉬워집니다. 예를 들어 파일을 열고 초기화 하기 위해 파스칼의 다른 객체처럼 *Create* 생성자를 사용하겠지만, 파일을 사용할 때 구조적 방법에서는 *AssignFile*, *Rewrite*, *Reset*, *Append* 프로시저로 초기화 합니다.

4.6 TFileStream을 사용한 파일 복사

이 예제에서는 *TFileStream* 형식을 사용하여 파일을 복사하고자 합니다.

이를 수행하려면 새 프로그램을 만들고 메인 폼에 이 구성 요소를 놓습니다.

TButton, TOpenDialog, TSaveDialog

버튼의 *OnClick* 이벤트에 대해 다음 코드를 작성합니다.

```
procedure TfmMain.Button1Click(Sender: TObject);
var
    SourceF, DestF: TFileStream;
    Buf: array [0 .. 1023] of Byte;
    NumRead: Integer;
begin
    if OpenDialog1.Execute and SaveDialog1.Execute then
    begin
        SourceF:= TFileStream.Create(OpenDialog1.FileName, fmOpenRead);
        DestF:= TFileStream.Create(SaveDialog1.FileName, fmCreate);
        while SourceF.Position < SourceF.Size do
        begin
            NumRead:= SourceF.Read(Buf, SizeOf(Buf));
            DestF.Write(Buf, NumRead);
        end;
        SourceF.Free;
        DestF.Free;
        ShowMessage('Copy finished');
    end;
end;
```

이 방법은 객체지향 파일을 사용한다는 점을 제외하면 비형식적 파일을 사용하는 것과 매우 유사합니다.

또한 파일을 복사하는 더욱 간단한 방법을 사용할 수 있습니다.

```
procedure TfmMain.Button1Click(Sender: TObject);
var
    SourceF, DestF: TFileStream;
begin
    if OpenDialog1.Execute and SaveDialog1.Execute then
    begin
        SourceF:= TFileStream.Create(OpenDialog1.FileName, fmOpenRead);
        DestF:= TFileStream.Create(SaveDialog1.FileName, fmCreate);
        DestF.CopyFrom(SourceF, SourceF.Size);
        SourceF.Free;
        DestF.Free;
        ShowMessage('Copy finished');
    end;
end;
```

우리가 원본 파일의 크기(SourceF.size)를 *CopyFrom* 메서드에 넘겨주었기 때문에 이것은 전체 파일 내용을 대상 파일로 복사합니다.

4.7 상속

객체지향 프로그래밍 에서의 상속은 기존의 클래스로부터 새로운 클래스를 만들고 기존의 메서드와 속성을 물려받음을 의미합니다. 기존의 속성과 메서드를 물려받고 나면, 새 클래스에 새 메서드와 속성을 추가할 수 있습니다.

상속의 예제로서 이전 문자열 큐 클래스로부터 새로운 정수형 큐 클래스를 만들려고 합니다. 정수형 큐를 처음부터 완전히 새로 작성하는 대신에 문자열 큐로부터 상속할 수 있습니다.

문자열 큐로부터 상속하려면, 새 Unit을 추가하고 *uses* 절에 문자열 큐를 놓습니다. 이제 새로운 정수형 큐를 다음과 같이 선언합니다.

```
    TIntQueue = class(TQueue)
```

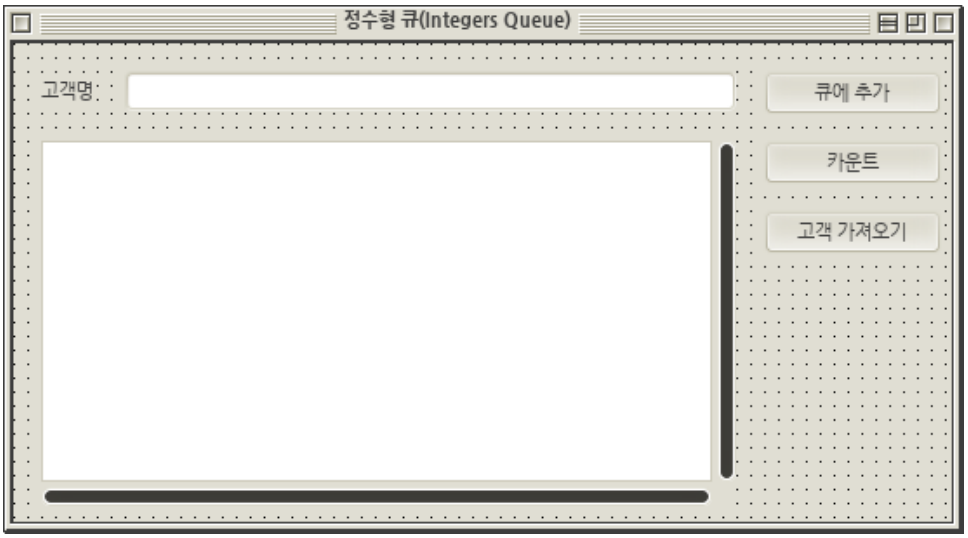
새 Unit의 이름은 *IntQueue*이고, 두 가지 새로운 메서드 *PutInt*와 *GetInt*를 도입하였습니다.

TIntQueue 클래스가 들어있는 Unit의 전체 코드입니다.

```
unit IntQueue;
// This unit contains TIntQueue class, which is inherits TQueue
// class and adds PutInt, GetInt methods to be used with
// Integer queue

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils, Queue;

type

    { TIntQueue }

    TIntQueue = class(TQueue)

    public
        function PutInt(AValue: Integer): Integer;
        function GetInt(var AValue: Integer): Boolean;
    end;

implementation

{ TIntQueue }

function TIntQueue.PutInt(AValue: Integer): Integer;
begin
    Result:= Put(IntToStr(AValue));
end;

function TIntQueue.GetInt(var AValue: Integer): Boolean;
var
    StrValue: string;
begin
    Result:= Get(StrValue);
    if Result then
        AValue:= StrToInt(StrValue);
end;

end.
```

참고로 부모 클래스 *TQueue*에 *Create*, *Destroy*, *Count* 메서드가 이미 존재하기 때문에 이들 메서드를 새로 작성하지 않았습니다.

새 정수형 큐 클래스를 사용하기 위해 새 프로그램을 만들고 이 구성요소들을 추가했습니다.



Unit의 코드입니다.

```
unit main;

{$mode objfpc}{$H+}

interface

uses
    Classes, SysUtils, FileUtil, LResources, Forms, Controls,
    Graphics, Dialogs, IntQueue, StdCtrls;

type

    { TfmMain }

    TfmMain = class(TForm)
        bbAdd: TButton;
        bbCount: TButton;
        bbGet: TButton;
        edCustomerID: TEdit;
        Label1: TLabel;
        Memo1: TMemo;
        procedure bbAddClick(Sender: TObject);
        procedure bbCountClick(Sender: TObject);
        procedure bbGetClick(Sender: TObject);
```

```
        procedure FormClose(Sender: TObject;
            var CloseAction: TCloseAction);
        procedure FormCreate(Sender: TObject);
    private
        { private declarations }
    public
        MyQueue: TIntQueue;
        { public declarations }
    end;

var
    fmMain: TfmMain;

implementation

{ TfmMain }

procedure TfmMain.FormCreate(Sender: TObject);
begin
    MyQueue:= TIntQueue.Create;
end;

procedure TfmMain.FormClose(Sender: TObject;
     var CloseAction: TCloseAction);
begin
    MyQueue.Free;
end;

procedure TfmMain.bbCountClick(Sender: TObject);
begin
    Memo1.Lines.Add('Queue length is: ' + IntToStr(MyQueue.Count));
end;

procedure TfmMain.bbAddClick(Sender: TObject);
var
    APosition: Integer;
begin
    APosition:= MyQueue.PutInt(StrToInt(edCustomerID.Text));
    Memo1.Lines.Add(edCustomerID.Text + ' has been added as # '
        + IntToStr(APosition + 1));
end;

procedure TfmMain.bbGetClick(Sender: TObject);
var
    ACustomerID: Integer;
begin
    if MyQueue.GetInt(ACustomerID) then
    begin
```

```
    Memo1.Lines.Add('Got: Customer ID : ' + IntToStr(ACustomerID) +
        ' from the queue');
    end
    else
        Memo1.Lines.Add('Queue is empty');
end;

initialization
    {$I main.lrs}

end.
```

참고로 *TQueue*의 속성과 메서드 그리고 추가적으로 *TIntQueue*의 속성과 메서드를 사용했습니다.

이 경우 초기 *TQueue*를 기반 클래스 또는 부모 *클래스*라고 부르며, 새 클래스를 *자식 클래스*라고 부릅니다.

새 클래스를 만드는 대신에 문자열 큐 Unit을 수정하고 정수형을 다루기 위해 *IntPut*과 *IntGet*을 추가할 수 있었지만, 상속을 표현하기 위해 새로운 *TIntQueue* 클래스를 만들었습니다. 또 다른 이유들이 있습니다. 라자루스의 *ppu*와 델파이의 *.dcu*와 같은 이미 컴파일한 Unit 파일만 가지고 있을 때와 같이 *TQueue*의 원본 소스 코드를 가지고 있지 못했다고 가정해봅니다. 이 경우 소스 코드를 볼 수 없고 물론 수정할 수도 없습니다. 상속은 이 큐에 대해 보다 기능적인 추가를 하기 위한 유일한 방법이 될 것입니다.

끝

Code.sd

2011. 06. 18