모듈 8

프로시저Procedure들

프로그램이 복잡해지지 않도록, 목적에 따라 프로시저를 만들 수 있게 됩니다.
그리고 이것을 활용할 수 있게 됩니다.
프로시저를 사용할 때 필요한 데이터를 넘기고 받을 수 있게 됩니다.

프로그래밍 언어의 연산자들을 안다고 해서, 편하게, 짧은 시간 만에, 오류가 거의 없이 프로그램을 작성할 수는 없다. 오히려 구조적structured 프로그래밍 이나 절차적procedural 프로그래밍의 기술과 원칙에 능숙해야 한다.

원칙들을 실제로 적용하여 얻게 되는 장점들은,

- 프로그램을 보다 편하게 작성할 수 있다.
- 이해하기 쉽고, 읽기 쉬운 프로그램을 만들 수 있다.
- 프로그램의 오류error를 더 재빠르게 찾아낼 수 있다.
- 프로그램을 더 재빠르게 변경하고 향상시킬 수 있다.

구조적structured 프로그래밍의 가장 중요한 원칙 중 하나는 탑-다운top-down 프로그래밍이다. 원칙은 어떤 프로그램이든 먼저 주요main 작업task을 하위작업subtask 몇 개로 나눈다. 그리고 나서, 각 하위작업을 다시 더 단순한 하위작업들 몇 개로 나눈다. 나누기를 계속하여 결국 아주 단순하고 구현하기 쉬운 작업이 되도록 한다.

쪼개진 하위작업을 델파이에서는 프로시저Procedure, 처리, 진행, 절차 라고 부른다.

이차방정식의 근을 구하는 문제를 풀어보자. 이차방정식의 계수 3개는 입력 받기로 한다.

Tip | 이차방정식을 모른다고 걱정하지 마세요. 본문에 공식이 적혀있으니, 그대로 하면 됩니다. 혹시라도 이차방정식을 더 알고 싶으면 구글 검색창에 '이차방정식 판별식' 등을 입력하면 공식부터 설명까지 다 나옵니다. 참고로 수학을 정말 잘하게 되면 탁월한 프로그래머가 될 수 있습니다. 이 책에서는 딱딱한 수학은 잊고 생각하는 수학을 즐기세요.

먼저, 새 폼을 하나 만들고 `frmQuadEq` 라고 이름을 붙인다. *Caption* 프로퍼티에는 '이차방정식 문제 풀이' 라고 적는다.

계수를 받을 텍스트박스 `edtA`, `edtB`, `edtC` 를 폼에 올린다. 이 텍스트박스들에 입력되는 값 3개를 계수로 한다. 텍스트박스의 *Text* 프로퍼티에서 읽으면 된다.

텍스트박스들 위에는 레이블 `lblCoefs` 를 두고 *Caption* 프로퍼티에 '이차방정식의 계수를 입력하세요' 라고 적는다.

레이블 `lblX1`, `lblX2` 를 올리고 여기에 이차방정식의 근 2개를 표시한다. 레이블 `lblNo` 도 추가하고 *Caption* 프로퍼티에 '근이 없습니다' 라고 적는다. 이 레이블 3개 모두 *Visible* 프로퍼티를 `False` 로 설정하여 프로그램이 시작될 때는 보이지 않도록 한다.

문제 풀기를 실행하도록 명령할 버튼 `btnFind`를 올린다. `btnFind` 버튼을 더블클릭하면 코드 편집기 화면이 표시된다. 이제 코드를 작성한다.

먼저 문제 풀이를 주석comment으로 적어보자. 델파이에서 주석은 { }대괄호 안이나, //빗금 두개 뒤에 적는다.

```
//이차방정식의 계수 A,B,C를 읽는다.
//판별식D를 계산한다.
//if D >= 0 then (D가 0보다 크거나 같으면)
    //방정식의 근 X1과 X2를 계산한다.
    //근을 표시한다.
//else (그렇지 않으면)
    //사용자에게 해법이 없다고 알린다.
```

위에 적은 풀이를 수행하려면 변수variable, 프로시저procedure, 함수function 가 필요하다. 델파이에서 한 문장으로 해결할 수 있는 경우라면, 그 문장 하나만 적기로 한다. 하지만 여러 문장으로 적어야 해결할 수 있다면 프로시저를 하나 만들고 그 안에 문장들을 넣는다.

위에서 주석으로 작성한 문제 풀이를 델파이로는 아래와 같이 적을 수 있다.

```
procedure TfrmQuadEq.btnFindClick(Sender: TObject);
var
   A, B, C, D, X1, X2: Real;
begin
   CoefInput(A, B, C); //계수 A,B,C를 읽는다.
   D:= Sqr(B)-4*A*C; //판별식D를 계산한다.
   if D>=0 then //(D가 0보다 크거나 같으면)
   begin
       Calc(A, B, D, X1, X2); //A,B,D를 이용해 근 x1, x2를 계산한다.
       Prn(X1, X2); //근 x1, x2를 출력한다
   end
   else //그렇지 않으면
       lblNo.Visible:=True; //사용자에게 해법이 없다고 알린다.
end;
```

위에 있는 프로시저의 본문 안에는 프로그램에서 실행해야 행위들 즉 프로시저Procedure, 처리, 진행, 절차 들이 들어있다. 그리고 각 프로시저 이름 뒤에 있는 괄호 안에 있는 값들은 프로시저의 파라미터parameter, 매개 변수라고 한다..

위 예문에서는 CoefInput, Calc, Prn 이 프로시저 이름이다. 델파이는 코드를 실행하면서, 델파이 문법의 키워드나 명령어가 아닌 이름을 만나게 되면, 그 이름을 가진 프로시저에게 통제권을 넘긴다. 이것을 프로시저 호출call 이라고 한다. 통제권를 받은 프로시저는 자신의 구문들을 실행하고 마치면 통제권을 다시 자신을 호출했던 루틴routine에게 넘겨준다. 그러면 그 루틴은 프로시저를 호출했던 문장의 다음에 있는 문장부터 진행을 계속한다.

프로시저 호출call 문장 구조

```
<프로시저 이름> (<프로시저에게 넘겨줄 파라미터들>);
```

프로시저를 호출하려면, 그 프로시저가 만들어져 있어야 한다. 즉, 우리는 호출하기 전에 먼저 구현Implementation 구역에 프로시저 정의definition 문장을 작성해야 한다.

프로시저 정의definition 문장 구조

```
procedure <프로시저 이름> (<프로시저가 사용하는 파라미터들의 이름과 타입>);
    <필요한 변수 등을 선언>
begin
    <구문들>
end;
```

위에서 보다시피, 프로시저를 호출 call 할 때에는 프로시저에게 넘겨줄 파라미터들을 넣어서 전달한다. 프로시저 정의 definition 구문에는 프로시저에서 사용하는 파라미터들의 이름과 타입을 명시한다. 프로시저를 호출할 때에는 프로시저 정의 구문에 명시된 파라미터들의 개수, 순서, 타입까지 모두 맞추어서 파라미터를 넘겨주어야 한다.

이제 손수 만드는 프로시저 즉 사용자 정의user-defined 프로시저를 만들어 보자. Prn 프로시저부터 시작하겠다.

Tip | 파라미터는 데이터를 다른 프로시저에게 전달하기 위해 사용합니다. 데이터를 전달하는 방식은 크게 2가지가 있습니다. 넘겨주는 데이터의 '값만 알려주는 값 파라미터 방식(Call By Value)' 과 데이터가 들어 있는 '그릇을 통채로 넣어서 전달하는 변수 파라미터 방식(Call By Reference)' 입니다. 파라미터를 정의할 때 (호출할 때가 아님!) var 키워드가 이름 앞에 붙으면 변수 파라미터이고, var 키워드가 없이 단지 이름과 타입만 있으면 값 파라미터입니다.

아래와 같이 프로시저 Prn에서는 Xf과 Xs라는 2개의 파라미터를 사용한다. 앞에 있는 버튼 클릭 이벤트 예문에서 Prn 프로시저를 호출하는 곳을 보면, 변수 X1과 X2를 프로시저 Prn에 정의된 파라미터 Xf과 Xs에 넣어서 전달한다. Xf과 Xs는 값value 파라미터이다. 따라서 호출하는 곳caller에서는 Prn 프로시저에게 값만 알려줄 뿐 되돌려 받지 않는다.

```
procedure Prn(Xf, Xs: Real);
begin
    frmQuadEq.lblX1.Visible:= True;
    frmQuadEq.lblX1.Caption:= 'x1='+FloatToStr(xf);
    frmQuadEq.lblX2.Visible:= True;
    frmQuadEq.lblX2.Caption:= 'x2='+FloatToStr(xs);
end;
```

위에서 파라미터로 근의 값들을 전달 받은 Prn 프로시저는 이 값들을 레이블에 넣어서 보여준다. 위 문장에서 frmQuadEq을 빼고 lblX1.Visible:= True 라고만 적으면 오류가 발생된다. 사용자 정의user-defined 프로시저 안에서는 사용할 컴포넌트의 전체 이름을 적어야 한다. (컴포넌트를 찾으려면 컴포넌트가 속해 있는 폼의 이름까지 모두 명시되어 있어야 한다. 대체로 프로그램에는 폼이 여러 개 들어 있기 때문이다)

계수들을 담는 프로시저를 작성해보자. 텍스트박스에서 입력된 숫자를 가져와 변수에 넣어야 한다. 프로시저를 호출할 때, 데이터를 전달할 뿐만 아니라 돌려 받기까지 하려면, 값이 아니라 변수를 파라미터로 넘겨주어야 한다. 이런 파라미터를 변수variable 파라미터parameter라고 한다. 값이 아니라 값을 담는 그릇 즉 변수이기 때문에 정의 구문에서 *var* 키워드를 앞에 붙인다. 호출하는 곳에서도 당연히 여기에 변수를 담아서 넘겨주어야 한다.

아래 구문에서 파라미터 k1, k2, k3 는 변수 파라미터이다. 따라서 CoefInput 프로시저를 호출 하는 곳에서는 자신이 가지고 있는 변수를 CoefInput 프로시저에게 넘겨주게 된다. 즉 CoefInput 프로시저의 k1, k2, k3 각각은 곧 호출한 곳 (즉 위의 버튼 클릭 이벤트핸들러)에서 넘겨준 변수 A, B, C를 프로시저 안에서 부르는 다른 이름이 되는 것이다. 그 결과 아래와 같이 k1, k2, k3의 값을 변경하게 되면, 호출한 곳에 있는 변수 즉 A, B, C의 값이 변경된다.

```
Procedure CoefInput(var k1, k2, k3: Real);
begin
   k1:= StrToFloat(frmQuadEq.edtA.Text);
   k2:= StrToFloat(frmQuadEq.edtB.Text);
   k3:= StrToFloat(frmQuadEq.edtC.Text);
end;
```

위에서 *StrToFloat* 함수가 사용되었다. 키보드로 입력되는 텍스트박스의 *Text* 프로퍼티는 *String* 타입이다. 하지만, 우리는 연산을 하기 위해 변수를 *Real* 타입으로 만들었기 때문에 *StrToFloat*를 사용하여 타입 변환conversion을 해주어야 한다. *StrToFloat* 함수를 적어 넣을 때에도 델파이 코드 완성 기능을 이용해 손쉽고 정확하게 입력할 수 있다.

이제 마지막 프로시저를 정의할 차례이다 —우리는 공식에 따라 방정식의 근들을 계산하고 찾아낸 근의 값들을 반환해야 한다. 따라서 이 프로시저에서 Xf 와 Xs 를 변수variable 파라미터로 받기로 한다. 하지만, 반환할 필요가 없는 판별식 값과 첫 번째, 두 번째 계수는 값value 파라미터로 넘겨받는다.

```
procedure Calc(k1, k2, dis: Real; var Xf, Xs: Real);
begin
   Xf:=(-k2+Sqrt(dis))/(2*k1);
   Xs:=(-k2-Sqrt(dis))/(2*k1);
end;
```

프로그램을 실행하고, 계수들을 입력한 후 버튼을 클릭하면, 폼 안에 레이블들이 표시된다. 다시 다른 계수들을 가지고 근을 구하려면 텍스트박스에 다른 계수를 입력한다. 하지만 화면에는 여전히 이전에 계산된 값들이 남아 있어서 보기가 좋지 않다. 이제 깨끗한 화면에서 새로 계산을 시작할 수 있도록, 화면을 정리해주는 프로시저를 하나 추가해보자. 이 프로시저는 파라미터가 필요없다. 프로시저의 이름은 `Init` 라고 하자. 화면을 깨끗이 하고 싶을 때면, 이 프로시저를 호출하면 된다.

```
procedure Init;
begin
    frmQuadEq.lblX1.Visible:=False;
    frmQuadEq.lblX2.Visible:=False;
    frmQuadEq.lblNo.Visible:=False;
end;
```

실습

Exercise 1.

프로그램을 하나 만들어서 변수 A의 값과 B의 값을 교환하고, 변수 C의 값과 D의 값을 교환해보자. 2개의 변수를 교환하는 프로시저를 정의하도록 한다.

Exercise 2.

삼각형의 세 변의 길이를 입력 받아서, 삼각형의 둘레와 면적을 계산하는 프로시저를 만들어보자.

Exercise 3.

반지 모양의 안쪽 지름 R1과 바깥쪽 지름 R2를 받아서 반지의 면적을 알아내자. 원의 면적을 계산하는 프로시저를 정의하도록 한다.

Exercise 4.

볼록사각형의 네 꼭지점들의 좌표를 받아서 그 볼록사각형의 면적을 계산해보자.