Prozedurdefinition

Parameter von Prozeduren können **statisch** oder **dynamisch** sein:

1) **statisch:**

```
VAR Res : INTEGER;
:
Res := Pot(2,3); (* Res = 8 *)
```

2) **dynamisch:**

```
VAR a,b,Res : INTEGER;
:
ReadInt(a); ReadInt(b);
:
Res := Pot(a,b);  (* Res = a^b *)
```


Prozedurdefinition

Prozedur wird im **Deklarationsteil** des Hauptprogramms definiert:

```
MODULE Taschenrechner;
FROM ...
VAR ...
PROCEDURE Pot(basis,exp:INTEGER) : INTEGER; (* Definition *)
 VAR ...
 BEGIN
   :    (* Implementierung der Prozedur *)
 END Pot;
BEGIN (* Hauptprogramm *)
   :
   ReadInt(x); ReadInt(y);
   WriteString ('Welche Operation (+,-,*,^) ?');
   Read(Op);
   CASE Op OF
        '+' : Erg := x+y;
    |   '^' : Erg := Pot(x,y); (* Prozeduraufruf *)
   :
END Taschenrechner.
```

Prozedurdefinition

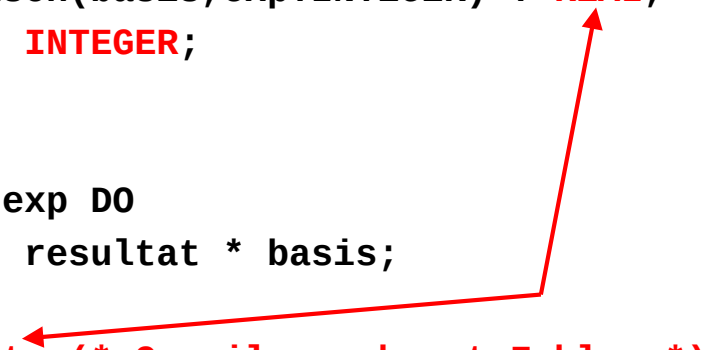
Definition einer Prozedur, die **`Pot` korrekt** implementiert:

```
PROCEDURE Pot(basis, exp : INTEGER) : INTEGER;
VAR i,resultat : INTEGER; (* lokale Variablen *)
BEGIN
  resultat := 1;
  FOR i := 1 TO exp DO
      resultat := resultat * basis;
  END;
  RETURN resultat; (* Rückgabe des
                      berechneten Wertes *)
END Pot;
```


Prozedurdefinition

Definition der Prozedur **`Pot_falsch`**, die einen **Typfehler** (type error) zur Übersetzungszeit zur Folge hat:

```
PROCEDURE Pot_falsch(basis,exp:INTEGER) : REAL;
 VAR i,resultat : INTEGER;
 BEGIN
   resultat := 1;
   FOR i := 1 TO exp DO
      resultat := resultat * basis;
   END;
   RETURN resultat; (* Compiler erkennt Fehler *)
 END Pot_falsch;
```



Prozedurdefinition

Das komplette Hauptprogramm:

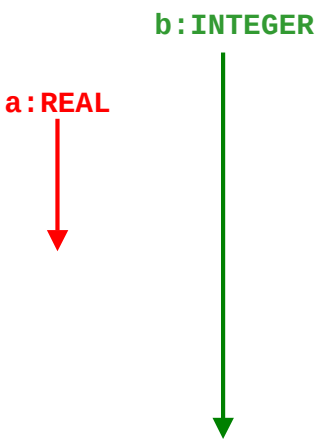
```
MODULE Taschenrechner;
FROM ...
VAR ...
PROCEDURE Pot(x,y:INTEGER) : INTEGER;
 VAR i,resultat : INTEGER;
 BEGIN
   :           (* Implementierung siehe oben *)
 END Pot;
BEGIN
   WriteString('Welche Operation (+,-,*,^) ?');
   Read(Op);
   CASE Op OF
        '+' : Erg := x+y;
        :
   |    '^' : Erg := Pot(x,y);
   END;
:
END Taschenrechner.
```


Gültigkeitsbereich (scope)

Variablen werden in **globale** und **lokale** Variablen unterteilt.

- **globale** Variablen gelten im **gesamten Programmbereich**
- **lokale** Variablen gelten nur **innerhalb einer Prozedur**

```
MODULE Scoping;
VAR b:INTEGER; (* global *)
PROCEDURE Local;
     VAR a:REAL; (* lokal *)
     BEGIN
       a:= 7.0; WriteReal(a);
       WriteInt(b);
     END Local;
BEGIN
     b:= 3;  WriteInt(b);
     Local;
     WriteInt(b);
END Scoping.
```

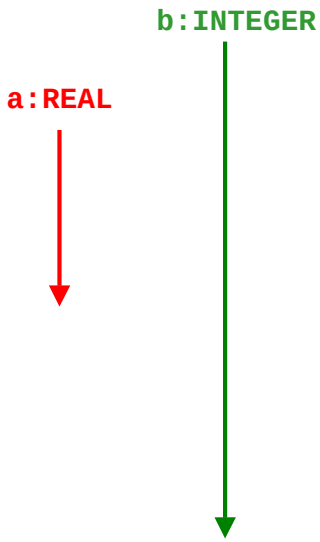


Ausgabe: Hauptprogramm: 3
Prozedur: 7.0 3
Hauptprogramm: 3

Gültigkeitsbereich (scope)

Globale Variablen können innerhalb einer Prozedur **verändert** werden.

```
MODULE Scoping;
VAR b:INTEGER; (* global *)
PROCEDURE Local;
     VAR a:REAL; (* lokal *)
     BEGIN
            a:= 7.0;
            b:= 2;
            WriteReal(a);
            WriteInt(b);
     END Local;
BEGIN
     b:= 3;
     WriteInt(b);
     Local;
     WriteInt(b);
END Scoping.
```



Gültigkeitsbereich (scope)

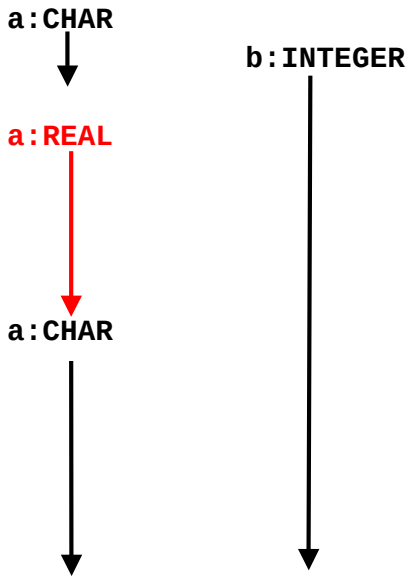
Ablauf des obigen Programms Scoping:

```
MODULE Scoping;
VAR b:INTEGER; (* global *)
PROCEDURE Local;
     VAR a:REAL; (* lokal *)
     BEGIN
            a:= 7.0; b:= 2;
            WriteReal(a); WriteInt(b);
     END Local;
BEGIN
     b:= 3;          Globale Zuweisung b:= 3
     WriteInt(b);    Ausgabe von b = 3
     Local;          Prozeduraufruf:
                       Zuweisung an die lokale Variable a:= 7.0
                       Änderung der globalen Variablen b auf b:= 2
                       lokale Ausgabe von a = 7.0 und b = 2
     WriteInt(b);    globale Ausgabe der geänderten Variable b = 2
END Scoping.
```

Gültigkeitsbereich (scope)

Hat eine lokale Variable **denselben Namen** wie eine globale,
so ist **nur** die **lokale** Variable **sichtbar**.
Die **gleichnamige globale** Variable ist solange **„verdeckt“**.

```
MODULE Scoping2;
 VAR    a : CHAR;
        b : INTEGER;
  PROCEDURE Local;
   VAR a : REAL;
   BEGIN
     a:= 7.0; WriteReal(a);
     b:= 2; WriteInt(b);
  END Local;
 BEGIN
   a:= 'X'; WriteChar(a);
   b:= 3; WriteInt(b);
   Local;
   WriteChar(a);
   WriteInt(b);
END Scoping2.
```



Gültigkeitsbereich (scope)

Ablauf des obigen Programms Scoping2:

```
MODULE Scoping2;
 VAR    a : CHAR;
        b : INTEGER;
  PROCEDURE Local;
   VAR a : REAL;
   BEGIN
     a:= 7.0; WriteReal(a);
     b:= 2; WriteInt(b);
  END Local;
 BEGIN
   a:= 'X'; WriteChar(a);
   b:= 3; WriteInt(b);
   Local;


   WriteChar(a);
   WriteInt(b);
END Scoping2.
```

Globale Zuweisung und Ausgabe von a:= ‘X‘
Globale Zuweisung und Ausgabe von b:= 3
Prozeduraufruf:
Zuweisung und Ausgabe der **lokalen** Variable **a:= 7.0**
Änderung und Ausgabe der **globalen** Variable b:= 2
Ausgabe der **globalen** Variable **a = ‘X‘**
Ausgabe der (geänderten) **globalen** Variable b = 2

Gültigkeitsbereich

In Modula-2 gelten folgende Scoping-Regeln:

1) Ein Bezeichner ist innerhalb der Prozedur, in der er definiert wurde und allen eingeschlossenen Prozeduren gültig. Ausnahmen werden durch Punkt 2 festgelegt. (Allgemeine Definitionsregel)

2) Wenn ein Bezeichner, der in einer Prozedur P deklariert worden ist, in einer inneren Prozedur Q redefiniert wird, sind die Prozedur Q und alle eingeschlossenen Prozeduren vom Gültigkeitsbereich, wie er in P definiert worden ist, ausgeschlossen. (Lokale Redefinition)

3) Standard-Bezeichner von Modula-2 werden betrachtet, als wären sie in einer imaginären Prozedur definiert, die jede Prozedur einschließt. (Standard-Bezeichner)

Gültigkeitsbereich

Es ist prinzipiell möglich innerhalb von Prozeduren sowohl lokale, als auch globale Variablen zu verwenden.

Aber:

Programmierhinweis:

In Unterprogrammen (Prozeduren) sollten nur Namen benutzt werden, die entweder als Parameter übergeben, oder lokal deklariert worden sind.

Vorteile:

- Ein solches Programm ist leichter verständlich und enthält keine **Seiteneffekte**. (reduced possibility of errors)
- Da überflüssige Variablen nach der Prozedurbeendigung ungültig werden, kann deren Speicherplatz freigegeben werden. (storage minimization)

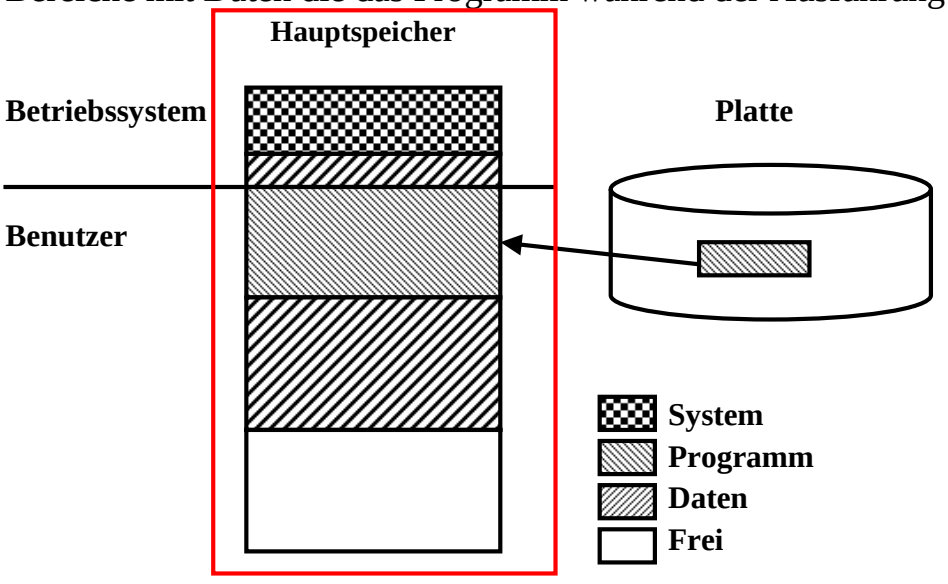
Gültigkeitsbereich

Die (unter Umständen ungewollte) Modifikation einer nicht-lokalen Umgebung wird allgemein als **Seiteneffekt** bezeichnet.

```
MODULE Seiteneffekt;
FROM InOut IMPORT WriteString,WriteInt;
VAR x: INTEGER;
   PROCEDURE f:INTEGER;
   BEGIN
     x:=2*x;  (* x wird mit 2 multipliziert und *)
              (* das Ergebnis wird in x gespeichert *)
     RETURN x;
   END f;
BEGIN
  x:=2;
  WriteInt(x);  (* x=2 *)
  WriteString('2 mal x:');
  WriteInt(f);  (* hier ändert sich x *)
  WriteInt(x);  (* x=4 *)
END Seiteneffekt;
```


Rückblick Speicherverwaltung

- Wird ein Programm gestartet, so wird das Programm vom Betriebssystem von der Festplatte in den Hauptspeicher geladen.
- Der Hauptspeicher enthält Bereiche für das Programm, sowie auch Bereiche mit Daten die das Programm während der Ausführung benötigt



Prozedurdefinition

Parameter von Prozeduren können **statisch** oder **dynamisch** sein:

1) **statisch:**

```
VAR Res : INTEGER;
:
Res := Pot(2,3); (* Res = 8 *)
```

2) **dynamisch:**

```
VAR a,b,Res : INTEGER;
:
ReadInt(a); ReadInt(b);
:
Res := Pot(a,b);  (* Res = a^b *)
```


Prozedurdefinition

Prozedur wird im **Deklarationsteil** des Hauptprogramms definiert:

```
MODULE Taschenrechner;
FROM ...
VAR ...
PROCEDURE Pot(basis,exp:INTEGER) : INTEGER; (* Definition *)
 VAR ...
 BEGIN
   :    (* Implementierung der Prozedur *)
 END Pot;
BEGIN (* Hauptprogramm *)
   :
   ReadInt(x); ReadInt(y);
   WriteString ('Welche Operation (+,-,*,^) ?');
   Read(Op);
   CASE Op OF
        '+' : Erg := x+y;
    |   '^' : Erg := Pot(x,y); (* Prozeduraufruf *)
   :
END Taschenrechner.
```

Prozedurdefinition

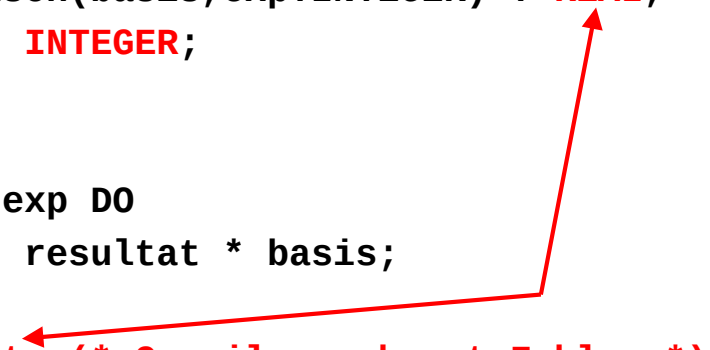
Definition einer Prozedur, die **`Pot` korrekt** implementiert:

```
PROCEDURE Pot(basis, exp : INTEGER) : INTEGER;
VAR i,resultat : INTEGER; (* lokale Variablen *)
BEGIN
  resultat := 1;
  FOR i := 1 TO exp DO
      resultat := resultat * basis;
  END;
  RETURN resultat; (* Rückgabe des
                      berechneten Wertes *)
END Pot;
```


Prozedurdefinition

Definition der Prozedur **`Pot_falsch`**, die einen **Typfehler** (type error) zur Übersetzungszeit zur Folge hat:

```
PROCEDURE Pot_falsch(basis,exp:INTEGER) : REAL;
 VAR i,resultat : INTEGER;
 BEGIN
   resultat := 1;
   FOR i := 1 TO exp DO
      resultat := resultat * basis;
   END;
   RETURN resultat; (* Compiler erkennt Fehler *)
 END Pot_falsch;
```



Prozedurdefinition

Das komplette Hauptprogramm:

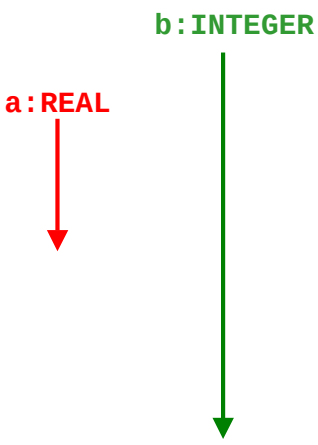
```
MODULE Taschenrechner;
FROM ...
VAR ...
PROCEDURE Pot(x,y:INTEGER) : INTEGER;
 VAR i,resultat : INTEGER;
 BEGIN
   :           (* Implementierung siehe oben *)
 END Pot;
BEGIN
   WriteString('Welche Operation (+,-,*,^) ?');
   Read(Op);
   CASE Op OF
        '+' : Erg := x+y;
        :
   |    '^' : Erg := Pot(x,y);
   END;
:
END Taschenrechner.
```


Gültigkeitsbereich (scope)

Variablen werden in **globale** und **lokale** Variablen unterteilt.

- **globale** Variablen gelten im **gesamten Programmbereich**
- **lokale** Variablen gelten nur **innerhalb einer Prozedur**

```
MODULE Scoping;
VAR b:INTEGER; (* global *)
PROCEDURE Local;
     VAR a:REAL; (* lokal *)
     BEGIN
       a:= 7.0; WriteReal(a);
       WriteInt(b);
     END Local;
BEGIN
     b:= 3;  WriteInt(b);
     Local;
     WriteInt(b);
END Scoping.
```

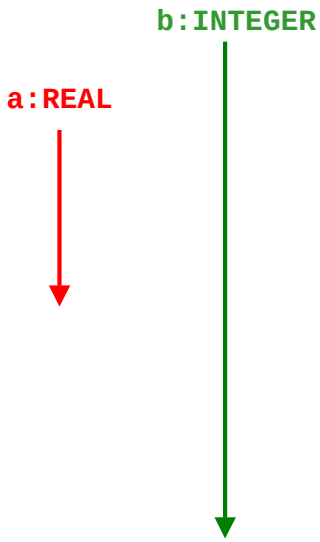


Ausgabe: Hauptprogramm: 3
Prozedur: 7.0 3
Hauptprogramm: 3

Gültigkeitsbereich (scope)

Globale Variablen können innerhalb einer Prozedur **verändert** werden.

```
MODULE Scoping;
VAR b:INTEGER; (* global *)
PROCEDURE Local;
     VAR a:REAL; (* lokal *)
     BEGIN
            a:= 7.0;
            b:= 2;
            WriteReal(a);
            WriteInt(b);
     END Local;
BEGIN
     b:= 3;
     WriteInt(b);
     Local;
     WriteInt(b);
END Scoping.
```



Gültigkeitsbereich (scope)

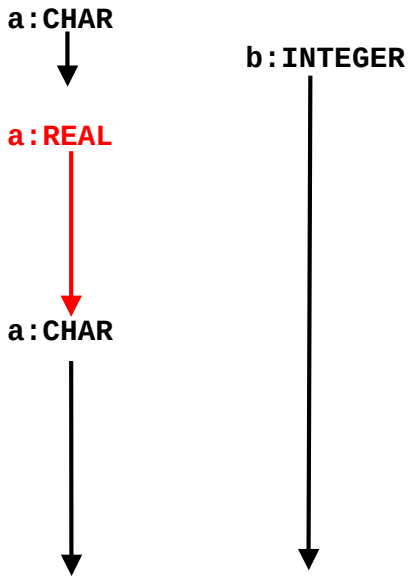
Ablauf des obigen Programms Scoping:

```
MODULE Scoping;
VAR b:INTEGER; (* global *)
PROCEDURE Local;
     VAR a:REAL; (* lokal *)
     BEGIN
            a:= 7.0; b:= 2;
            WriteReal(a); WriteInt(b);
     END Local;
BEGIN
     b:= 3;          Globale Zuweisung b:= 3
     WriteInt(b);    Ausgabe von b = 3
     Local;          Prozeduraufruf:
                       Zuweisung an die lokale Variable a:= 7.0
                       Änderung der globalen Variablen b auf b:= 2
                       lokale Ausgabe von a = 7.0 und b = 2
     WriteInt(b);    globale Ausgabe der geänderten Variable b = 2
END Scoping.
```

Gültigkeitsbereich (scope)

Hat eine lokale Variable **denselben Namen** wie eine globale,
so ist **nur** die **lokale** Variable **sichtbar**.
Die **gleichnamige globale** Variable ist solange **„verdeckt“**.

```
MODULE Scoping2;
 VAR    a : CHAR;
        b : INTEGER;
  PROCEDURE Local;
   VAR a : REAL;
   BEGIN
     a:= 7.0; WriteReal(a);
     b:= 2; WriteInt(b);
  END Local;
 BEGIN
   a:= 'X'; WriteChar(a);
   b:= 3; WriteInt(b);
   Local;
   WriteChar(a);
   WriteInt(b);
END Scoping2.
```



Gültigkeitsbereich (scope)

Ablauf des obigen Programms Scoping2:

```
MODULE Scoping2;
 VAR    a : CHAR;
        b : INTEGER;
  PROCEDURE Local;
   VAR a : REAL;
   BEGIN
     a:= 7.0; WriteReal(a);
     b:= 2; WriteInt(b);
  END Local;
 BEGIN
   a:= 'X'; WriteChar(a);
   b:= 3; WriteInt(b);
   Local;


   WriteChar(a);
   WriteInt(b);
END Scoping2.
```

Globale Zuweisung und Ausgabe von a:= ‘X‘
Globale Zuweisung und Ausgabe von b:= 3
Prozeduraufruf:
Zuweisung und Ausgabe der **lokalen** Variable **a:= 7.0**
Änderung und Ausgabe der **globalen** Variable b:= 2
Ausgabe der **globalen** Variable **a = ‘X‘**
Ausgabe der (geänderten) **globalen** Variable b = 2

Gültigkeitsbereich

In Modula-2 gelten folgende Scoping-Regeln:

1) Ein Bezeichner ist innerhalb der Prozedur, in der er definiert wurde und allen eingeschlossenen Prozeduren gültig. Ausnahmen werden durch Punkt 2 festgelegt. (Allgemeine Definitionsregel)

2) Wenn ein Bezeichner, der in einer Prozedur P deklariert worden ist, in einer inneren Prozedur Q redefiniert wird, sind die Prozedur Q und alle eingeschlossenen Prozeduren vom Gültigkeitsbereich, wie er in P definiert worden ist, ausgeschlossen. (Lokale Redefinition)

3) Standard-Bezeichner von Modula-2 werden betrachtet, als wären sie in einer imaginären Prozedur definiert, die jede Prozedur einschließt. (Standard-Bezeichner)

Gültigkeitsbereich

Es ist prinzipiell möglich innerhalb von Prozeduren sowohl lokale, als auch globale Variablen zu verwenden.

Aber:

Programmierhinweis:

In Unterprogrammen (Prozeduren) sollten nur Namen benutzt werden, die entweder als Parameter übergeben, oder lokal deklariert worden sind.

Vorteile:

- Ein solches Programm ist leichter verständlich und enthält keine **Seiteneffekte**. (reduced possibility of errors)
- Da überflüssige Variablen nach der Prozedurbeendigung ungültig werden, kann deren Speicherplatz freigegeben werden. (storage minimization)

Gültigkeitsbereich

Die (unter Umständen ungewollte) Modifikation einer nicht-lokalen Umgebung wird allgemein als **Seiteneffekt** bezeichnet.

```
MODULE Seiteneffekt;
FROM InOut IMPORT WriteString,WriteInt;
VAR x: INTEGER;
   PROCEDURE f:INTEGER;
   BEGIN
     x:=2*x;  (* x wird mit 2 multipliziert und *)
              (* das Ergebnis wird in x gespeichert *)
     RETURN x;
   END f;
BEGIN
  x:=2;
  WriteInt(x);  (* x=2 *)
  WriteString('2 mal x:');
  WriteInt(f);  (* hier ändert sich x *)
  WriteInt(x);  (* x=4 *)
END Seiteneffekt;
```


Rückblick Speicherverwaltung

- Wird ein Programm gestartet, so wird das Programm vom Betriebssystem von der Festplatte in den Hauptspeicher geladen.
- Der Hauptspeicher enthält Bereiche für das Programm, sowie auch Bereiche mit Daten die das Programm während der Ausführung benötigt