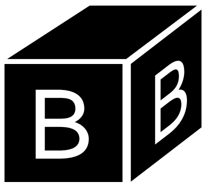
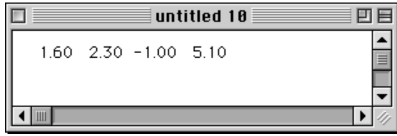


Chapter *15*

One-Dimensional Arrays



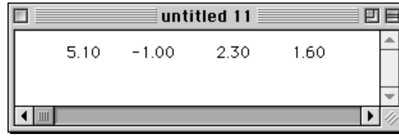


Figure 15.1
The input and output for the procedure of Figure 15.2.

```
MODULE Pbox15A;
    IMPORT TextModels, TextViews, Views, TextControllers, PboxMappers;

    TYPE
        Real4 = ARRAY 4 OF REAL;

    PROCEDURE ReverseReals*;
        VAR
            mdIn: TextModels.Model;
            cn: TextControllers.Controller;
            sc: PboxMappers.Scanner;
            list: Real4;
            i: INTEGER;
            mdOut: TextModels.Model;
            vw: TextViews.View;
            fm: PboxMappers.Formatter;
```

Figure 15.2
A program to reverse four real values in the focus window.

```
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            mdIn := cn.text;
            sc.ConnectTo(mdIn);
            FOR i := 0 TO 3 DO
                sc.ScanReal(list[i])
            END;
            mdOut := TextModels.dir.New();
            fm.ConnectTo(mdOut);
            FOR i := 3 TO 0 BY -1 DO
                fm.WriteReal(list[i], 8, 2)
            END;
            vw := TextViews.dir.New(mdOut);
            Views.OpenView(vw)
        END
    END ReverseReals;

END Pbox15A.
```

i	
list[0]	
list[1]	
list[2]	
list[3]	

(a) Initially.

i	0
list[0]	1.60
list[1]	
list[2]	
list[3]	

(b) After first scan.

i	1
list[0]	1.60
list[1]	2.30
list[2]	
list[3]	

(c) After second scan.

Figure 15.3
A trace of procedure ReverseReals in Figure 15.2.

i	2
list[0]	1.60
list[1]	2.30
list[2]	-1.00
list[3]	

(d) After third scan.

i	3
list[0]	1.60
list[1]	2.30
list[2]	-1.00
list[3]	5.10

(e) After fourth scan.

i	3
list[0]	1.60
list[1]	2.30
list[2]	-1.00
list[3]	5.10

(f) Output list[3].

i	2
list[0]	1.60
list[1]	2.30
list[2]	-1.00
list[3]	5.10

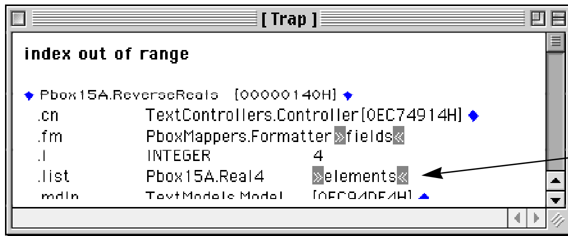
(g) Output list[2].

i	1
list[0]	1.60
list[1]	2.30
list[2]	-1.00
list[3]	5.10

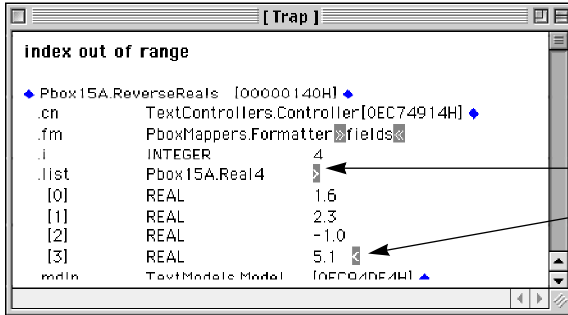
(h) Output list[1].

i	0
list[0]	1.60
list[1]	2.30
list[2]	-1.00
list[3]	5.10

(i) Output list[0].



(**a**) The values of list are hidden in the fold.



(**b**) Expanding the fold to see the values of list.

Figure 15.4
The trap window when your index is out of range.

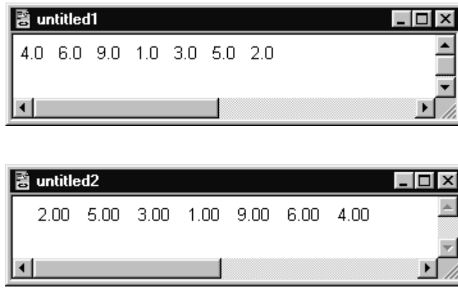


Figure 15.5
The input and output for the procedure of Figure 15.6.

PROCEDURE (VAR s: Scanner) **ScanRealVector** (OUT v: ARRAY OF REAL; OUT numItm: INTEGER), NEW
Pre
s is connected to a text model.   20
Sequences of characters scanned represent in-range real or integer values.   21
Number of values in text model <= LEN(v).   Index out of range.
Post
v gets all the values scanned up to the end of the text model to which s is connected.
numItm gets the number of integer values scanned.
The values are stored at v[0..numItm - 1].

```
MODULE Pbox15B;
   IMPORT TextModels, TextViews, Views, TextControllers, PboxMappers;

   TYPE
      Real1024 = ARRAY 1024 OF REAL;

   PROCEDURE ReverseReals*;
      VAR
         mdIn: TextModels.Model;
         cn: TextControllers.Controller;
         sc: PboxMappers.Scanner;
         list: Real1024;
         numItems: INTEGER;
         i: INTEGER;
         mdOut: TextModels.Model;
         vw: TextViews.View;
         fm: PboxMappers.Formatter;
```

Figure 15.6
A program to reverse any number of real values.

```
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            mdIn := cn.text;
            sc.ConnectTo(mdIn);
            sc.ScanRealVector(list, numItems);
            mdOut := TextModels.dir.New();
            fm.ConnectTo(mdOut);
            FOR i := numItems - 1 TO 0 BY -1 DO
                fm.WriteReal(list[i], 6, 2)
            END;
            vw := TextViews.dir.New(mdOut);
            Views.OpenView(vw)
        END
    END ReverseReals;

END Pbox15B.
```

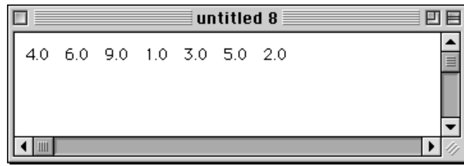
- *Step 1*—Write some specific initial values for the array in a trace.
- *Step 2*—Perform the manipulation by changing the values in the trace, one at a time.
- *Step 3*—For each change, write a specific assignment statement that will produce the change.
- *Step 4*—Discover a pattern in the indices of the assignment statements you wrote. Generalize from the specific statements to a loop containing arrays with variables in the subscripts.

```
MODULE Pbox15C;
    IMPORT TextModels, TextViews, Views, TextControllers, PboxMappers;

    TYPE
        Real1024 = ARRAY 1024 OF REAL;

    PROCEDURE RotateLeft (VAR v: ARRAY OF REAL; numItm: INTEGER);
        VAR
            i: INTEGER;
            temp: REAL;
    BEGIN
        ASSERT((0 <= numItm) & (numItm <= LEN(v)), 20);
        IF numItm > 1 THEN
            temp := v[0];
            FOR i := 0 TO numItm - 2 DO
                v[i] := v[i + 1]
            END;
            v[numItm - 1] := temp
        END
    END RotateLeft;
```

Figure 15.7
A program with a procedure to rotate the elements in an array.

```
PROCEDURE ProcessRotation*;
    VAR
        mdIn: TextModels.Model;
        cn: TextControllers.Controller;
        sc: PboxMappers.Scanner;
        list: Real1024;
        numItems: INTEGER;
        mdOut: TextModels.Model;
        vw: TextViews.View;
        fm: PboxMappers.Formatter;
```

```
    BEGIN
        cn := TextControllers.Focus();
        IF cn # NIL THEN
            mdIn := cn.text;
            sc.ConnectTo(mdIn);
            sc.ScanRealVector(list, numItems);
            mdOut := TextModels.dir.New();
            fm.ConnectTo(mdOut);
            fm.WriteRealVector(list, numItems, 6, 2);
            RotateLeft(list, numItems);
            fm.WriteLn; fm.WriteLn;
            fm.WriteRealVector(list, numItems, 6, 2);
            vw := TextViews.dir.New(mdOut);
            Views.OpenView(vw)
        END
    END ProcessRotation;

END Pbox15C.
```



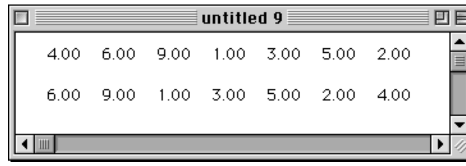
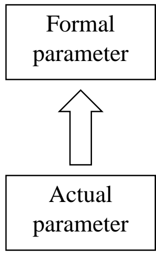
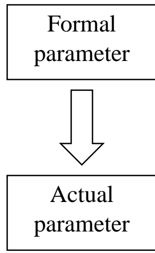


Figure 15.8
The input and output for the procedure of Figure 15.7.

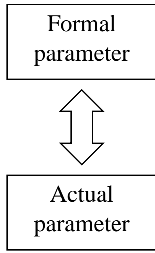
PROCEDURE (VAR f: Formatter) **WriteRealVector** (IN v: ARRAY OF REAL; numItm, minWidth, dec: INTEGER),
NEW
Pre
f is connected to a text model. 20
numItm <= LEN(v). Index out of range.
Post
The first numItm values of v are written to the text model to which f is connected,
each with a field width of minWidth and dec places past the decimal point. If minWidth
is too small to contain a value of v it expands to accommodate the value.



(a) Call by value and call by constant reference.



(b) Call by result.



(c) Call by reference.

Figure 15.9
The flow of information for four different calling mechanisms, including call by constant reference.

	Integers, reals, booleans, pointers, short arrays and short records	Long arrays and long records
Call by value Default	Common. Use when the actual parameter should *not* change. Actual parameter can be an expression.	Not common. Inefficient procedure call. Use call by constant reference instead.
Call by constant reference IN	Not common (illegal for all types except arrays and records). Inefficient procedure execution. Use call by value instead.	Common. Use when the actual parameter should *not* change Actual parameter must be a variable.
Call by result OUT	Common. Use when the actual parameter *should* change and its initial value is *undefined*. Actual parameter must be a variable.	
Call by reference VAR	Common. Use when the actual parameter *should* change and its initial value is *defined*. Actual parameter must be a variable.	

Figure 15.10
Guidelines for using the four parameter calling mechanisms.

```
PROCEDURE Maximum (IN v: ARRAY OF REAL; numItm: INTEGER): REAL;
    VAR
        i: INTEGER;
        largest: REAL;
BEGIN
    ASSERT((0 < numItm) & (numItm <= LEN(v)), 20);
    largest := v[0];
    FOR i := 1 TO numItm - 1 DO
        IF v[i] > largest THEN
            largest := v[i]
        END
    END;
    RETURN largest
END Maximum;
```

Figure 15.11
A function that returns the largest element in an array.

```
PROCEDURE LargestLast (VAR v: ARRAY OF REAL; numItm: INTEGER);
   VAR
      i, indexMax: INTEGER;
      temp: REAL;
BEGIN
   ASSERT((0 <= numItm) & (numItm <= LEN(v)), 20);
   IF numItm > 1 THEN
      indexMax := 0;
      FOR i := 1 TO numItm - 1 DO
         IF v[i] > v[indexMax] THEN
            indexMax := i
         END
      END;
      temp := v[numItm - 1];
      v[numItm - 1] := v[indexMax];
      v[indexMax] := temp
   END
END LargestLast;
```

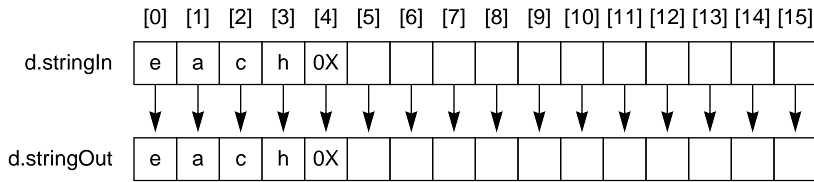
Figure 15.12
A procedure that exchanges the largest element in an array with the last element.

```
PROCEDURE Initialize (OUT v: ARRAY OF INTEGER; numItm: INTEGER);
    VAR
        i: INTEGER;
BEGIN
    ASSERT((0 <= numItm) & (numItm <= LEN(v)), 20);
    FOR i := 0 TO numItm - 1 DO
        v[i] := numItm - i - 1
    END
END Initialize;
```

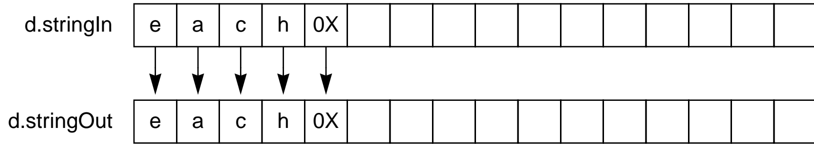
Figure 15.13
A procedure to initialize the elements of an array to a decreasing sequence.

```
PROCEDURE Initialize (OUT v: ARRAY OF INTEGER; numItm: INTEGER);
    VAR
        i, j: INTEGER;
BEGIN
    ASSERT((0 <= numItm) & (numItm <= LEN(v)), 20);
    j := numItm - 1;
    FOR i := 0 TO numItm - 1 DO
        v[i] := j;
        DEC(j)
    END;
END Initialize;
```

Figure 15.14
A procedure that performs the identical processing to that in Figure 15.13.



(a) d.stringOut := d.stringIn



(b) d.stringOut := d.stringIn$

Figure 15.15
The effect of the $ when specifying an array of characters.

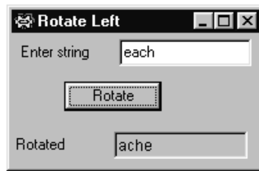


Figure 15.16
The dialog box for a program that rotates an array of characters to the left.

```
MODULE Pbox15D;
    IMPORT Dialog, PboxStrings;
    TYPE
        String16 = ARRAY 16 OF CHAR;
    VAR
        d*: RECORD
            stringIn*: String16;
            stringOut-: String16
        END;
```

Figure 15.17
A program with a procedure to rotate the elements in an array.

```
    PROCEDURE RotateLeft (VAR str: ARRAY OF CHAR);
        VAR
            i: INTEGER;
            temp: CHAR;
    BEGIN
        IF LEN(str$) > 1 THEN
            temp := str[0];
            FOR i := 0 TO LEN(str$) - 2 DO
                str[i] := str[i + 1]
            END;
            str[LEN(str$) - 1] := temp
        END
    END RotateLeft;

    PROCEDURE ProcessRotation*;
    BEGIN
        d.stringOut := d.stringIn$;
        RotateLeft(d.stringOut);
        Dialog.Update(d)
    END ProcessRotation;

BEGIN
    d.stringIn := "";
    d.stringOut := ""
END Pbox15D.
```

```
PROCEDURE RotateLeft (VAR str: ARRAY OF CHAR);
    VAR
        i: INTEGER;
        temp: CHAR;
BEGIN
    IF str[0] # 0X THEN
        temp := str[0];
        i := 0;
        WHILE str[i + 1] # 0X DO
            str[i] := str[i + 1];
            INC(i)
        END;
        str[i] := temp
    END
END RotateLeft;
```

Figure 15.18
A more efficient version of RotateLeft.

```
FOR i := 0 TO numItms - 2 DO
    v[i] := v[i + 1]
END
```

$i := 0$;
do $i \le n - 2 \rightarrow$
 $v[i] := v[i + 1]$;
 $i := i + 1$
od

```
BEGIN
    ASSERT((0 <= numItm) & (numItm <= LEN(v)), 20);
    IF numItm > 1 THEN
        temp := v[0];
        FOR i := 0 TO numItm - 2 DO
            v[i] := v[i + 1]
        END;
        v[numItm - 1] := temp
    END
END RotateLeft;
```

```
temp := v[0];
FOR i := 0 TO numItm - 2 DO
    v[i] := v[i + 1]
END;
v[numItm - 1] := temp
```

```
temp := v[0];
FOR i := 0 TO numItm - 2 DO
    v[i] := v[i + 1]
END;
v[numItm - 1] := temp
```

$t := v[0]$;
$i := 0$;
do $i \le n - 2 \rightarrow$
    $v[i] := v[i + 1]$;
    $i := i + 1$
od;
$v[n - 1] := t$

$\{P\}S\{Q\}$

$\{1 < n \le len(v) \land (\forall i \mid 0 \le i < n : v[i] = \mathbf{v}[i])\}$
$v := ?$
$\{(\forall i \mid 0 \le i < n - 1 : v[i] = \mathbf{v}[i + 1]) \land v[n - 1] = \mathbf{v}[0]\}$

$\{1 < n \le len(v) \wedge (\forall i \mid 0 \le i < n : v[i] = \mathbf{v}[i])\}$
$t := v[0]$;
$i := 0$;
do $i \le n - 2 \rightarrow$
  $v[i] := v[i + 1]$;
  $i := i + 1$
od;
$v[n - 1] := t$
$\{(\forall i \mid 0 \le i < n - 1 : v[i] = \mathbf{v}[i + 1]) \wedge v[n - 1] = \mathbf{v}[0]\}$

$v = \mathtt{v}$

is an abbreviation for

$(\forall i \mid 0 \le i < n : v[i] = \mathtt{v}[i])$

$\{1 < n \leq len(v) \land v = \mathtt{V}\}$
$v := ?$
$\{(\forall i \mid 0 \leq i < n - 1 : v[i] = \mathtt{V}[i + 1]) \land v[n - 1] = \mathtt{V}[0]\}$

```
largest := v[0];
FOR i := 1 TO numItm - 1 DO
    IF v[i] > largest THEN
        largest := v[i]
    END
END
```

```
largest := v[0];
FOR i := 1 TO numItm - 1 DO
    IF v[i] > largest THEN
        largest := v[i]
    END
END
```

$\{0 < n \leq len(v)\}$

```
largest := v[0];
FOR i := 1 TO numItm - 1 DO
    IF v[i] > largest THEN
        largest := v[i]
    END
END
```

$\{0 < n \leq len(v)\}$
$g := ?$

```
largest := v[0];
FOR i := 1 TO numItm - 1 DO
    IF v[i] > largest THEN
        largest := v[i]
    END
END
```

$\{0 < n \le len(v)\}$
$g := ?$
$\{(\exists i \mid 0 \le i < n : g = v[i]) \wedge (\forall i \mid 0 \le i < n : g \ge v[i])\}$

$perm(a,b,f,l)$.

$(\forall i \mid f \le i \le l : (\exists j \mid f \le j \le l : a[i] = b[j]))$