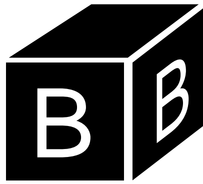


Chapter *4*

Variables

Every Component Pascal variable has three attributes:

The three attributes of a variable

- A name
- A type
- A value

A variable's name is an identifier determined arbitrarily by the programmer. A variable's type specifies the kind of values it can have. Variable names and types are declared in the declaration sequence, which must be placed before the executable statements of a procedure. Unlike a variable's name and type, a variable's value does not in general appear in a program listing. The value is contained in main memory during execution of the program.

Real variables

Figure 4.1 shows how to declare real variables in a program. The output of procedure Rectangle is:

```
The width is 3.6
The length is 12.4
```

```
MODULE Pbox04A;
   IMPORT StdLog;

   PROCEDURE Rectangle*;
   VAR
      width: REAL;
      length: REAL;
   BEGIN
      width := 3.6;
      length := 12.4;
      StdLog.String("The width is "); StdLog.Real(width); StdLog.Ln;
      StdLog.String("The length is "); StdLog.Real(length); StdLog.Ln
   END Rectangle;

END Pbox04A.
```

Figure 4.1
A procedure that sets the value of two real variables and outputs them to the Log.

The modules in Chapter 3 have names that begin with Hw99 to illustrate how you should name your modules if you are student number 99 in a class of many students. Beginning with this chapter, most modules will be named according to the chapter number of the book. Hence, the name of the module in Figure 4.1 is Pbox04A, where Pbox04 represents Chapter 4 of programming with BlackBox using *Computing Fundamentals*, and A is the first program in the chapter. Pbox04B will be the name of the next module in this chapter, and so on. If you are studying this book as part of a class, you should continue to use your assigned number with the convention you learned in Chapter 3.

As Figure 4.1 shows, the declaration sequence begins with the reserved word VAR and contains a list of all variables used in the procedure. width is the first variable's name, and REAL is its type. The type REAL means the variable's value will be a real number, with a fractional part indicated by a decimal point. The name and type of a variable are separated by a colon.

Variables of type REAL

Notice how semicolons are used in a declaration sequence. One of the EBNF alternatives for a declaration sequence is

VAR {VarDecl “ ; ”}

which shows that semicolons serve to terminate a variable declaration. They do not separate one variable declaration from the following variable declaration. The semicolon that terminates the variable declaration

```
length: REAL;
```

is necessary even though it occurs before BEGIN, which is not a statement.

Assignment statements

Unlike names and types, the values of the variables are usually not visible in the program listing (although they are in this program). Instead, they exist in main memory during program execution. An assignment statement sets the value of a variable. The assignment statement

```
width := 3.6;
```

sets the value of the variable width to 3.6. The := symbol is called the *assignment symbol*. You should read this statement in English as “width gets 3.6.” Do not say “width equals 3.6.” The equals symbol, =, has a different meaning in Component Pascal from the assignment symbol, :=.

The assignment symbol is pronounced “gets”.

The name of a variable must be on the left side of the assignment symbol, and an expression must be on the right side. A numeric value such as 3.6 is an example of a real expression.

Real output

Figure 3.4 shows the interface for the StdLog module. The specification for proce-

dure Real in the StdLog module is

```
PROCEDURE Real (x: REAL);
```

The parameter x in the interface is the formal parameter. It has type REAL. The types of the formal parameters tell you what is allowed in the actual parameters. In the program in Figure 4.1 the first call to StdLog.Real is

Formal parameters

```
StdLog.Real(width)
```

The actual parameter in the procedure call is width. Actual parameter width corresponds to formal parameter x. The procedure call adheres to the specification given in the interface, because the type of actual parameter width corresponds to the type of formal parameter x—both are REAL. The value for x that you give to procedure Real is the value that you want to print on the Log.

Actual parameters

Real expressions

The four real operations in Component Pascal are addition, subtraction, multiplication, and division, indicated symbolically by +, -, *, and / as summarized in Figure 4.2. They have the same precedence you are familiar with from algebra. The operators * and / have a higher precedence than + and -. When parentheses are present in the expression, the contents of the parentheses are evaluated first.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division

Figure 4.2
The real operators.

Example 4.1 Two examples of expressions and their evaluations without parentheses are

```
4.0 * 5.5 + 6.0          4.0 + 5.5 * 6.0
22.0 + 6.0               4.0 + 33.0
28.0                     37.0
```

The multiplication operation is performed first because it has higher precedence than addition. ▮

Example 4.2 An example with parentheses is

4.0 * (5.5 + 6.0)
4.0×11.5
46.0

The addition is performed before the multiplication because the addition is within parentheses. ■

Associative operators

An operator ρ is associative if $(a\ \rho\ b)\rho\ c \equiv a\ \rho(b\ \rho\ c)$. For example, + is associative, because $(a+b)+c \equiv a+(b+c)$. However, – is not associative, because it is not the case that $(a-b)-c \equiv a-(b-c)$. If two operators of the same precedence are adjacent, the evaluation is done from left to right. This rule makes a difference if an operator is not associative.

Left-to-right rule

Example 4.3 Two examples of the left-to-right rule are

11.5 - 3.0 - 4.5	25.0 / 10.0 / 5.0
8.5 – 4.5	2.5 / 5.0
4.0	0.5

Notice the difference that this rule makes in the results. If you first subtract 4.5 from 3.0 to get –1.5, and then subtract that from 11.5, you get 13.0, which is different from the correct value of 4.0. Similarly, if you first divide 10.0 by 5.0 to get 2.0, and then divide 25.0 by 2.0, you get 12.5, which is different from the correct value of 0.5. ■

```
MODULE Pbox04B;
    IMPORT StdLog;

    PROCEDURE Rectangle*;
    VAR
        width, length: REAL;
        area, perim: REAL;
    BEGIN
        width := 3.6;
        length := 12.4;
        StdLog.String("The width is "); StdLog.Real(width); StdLog.Ln;
        StdLog.String("The length is "); StdLog.Real(length); StdLog.Ln;
        area := width * length;
        perim := 2.0 * (width + length);
        StdLog.String("The area is "); StdLog.Real(area); StdLog.Ln;
        StdLog.String("The perimeter is "); StdLog.Real(perim); StdLog.Ln
    END Rectangle;

END Pbox04B.
```

Figure 4.3 Using real expressions in a program.

Figure 4.3 shows how to use a real expression in a complete program. The output on the Log of this program is

```
The width is 3.6
The length is 12.4
The area is 44.64
The perimeter is 32.0
```

Integer variables

Figure 4.4 shows how to declare an integer variable in a program. You should check the interface for module StdLog to see the specification for procedure Int. The output of the program is:

```
You have 39 cents in change.
```

```
MODULE Pbox04C;
   IMPORT StdLog;

   PROCEDURE Change*;
   VAR
      cents: INTEGER;
   BEGIN
      cents := 39;
      StdLog.String("You have "); StdLog.Int(cents);
      StdLog.String(" cents in change."); StdLog.Ln
   END Change;

END Pbox04C.
```

Figure 4.4
A procedure that sets the value of an integer variable and outputs it to the Log.

Computers store integer values in main memory differently from real values. To store an integer, the computer has two storage compartments—one for the sign of the number and one for its magnitude. However, to store a real value, the computer uses binary scientific notation with four storage compartments—one for the sign of the exponent (the power of 2), one for the exponent, one for the sign of the value, and one for the magnitude.

Because of this difference in the way the computer stores integer and real values, Component Pascal puts some restrictions on how you use them in a program. The procedure in Figure 4.5 illustrates the fact that you cannot assign a real value to an integer variable. The procedure has an assignment incompatibility error and will not compile.

You can, however, assign an integer value to a real variable. Component Pascal will convert the integer value to the corresponding equal real value before making the assignment.

Example 4.4 If you declare x to have type real then the assignment statement

```
x := 5
```

is legal even though 5 is an integer value. Component Pascal converts the integer value 5 to the real value 5.0 before making the assignment to x. ∎

```
MODULE Pbox04D;
    IMPORT StdLog;

    PROCEDURE Error*;
    VAR
        i: INTEGER;
    BEGIN
        i := 2.7;
        StdLog.String("The value of i is "); StdLog.Int(i); StdLog.Ln
    END Error;

END Pbox04D.
```

Figure 4.5
A procedure that tries to assign a real value to an integer variable. This procedure has a bug.

Integer expressions

Integer values, which do not have fractional parts, are used for counting whole objects. For example, if you need to keep track of the number of employees who work for your company, you could have a variable, numEmpl, of type integer whose value represents the number of workers the company has. numEmpl could never have a value like 234.6, because you cannot have 0.6 of an employee. Figure 4.6 summarizes the integer operations.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
DIV	Division
MOD	Modulo

Figure 4.6
The integer operators.

Addition, subtraction, and multiplication for integer values are similar to the same operations for real values, but division is different. In integer division, denoted by the operator DIV, a fractional part cannot be included in the result. Instead, the fractional part is discarded, or truncated.

Example 4.5 The real expression 14.0 / 3.0 evaluates to 4.667, but the integer expression 14 DIV 3 evaluates to 4. Further examples of DIV are:

15 DIV 3 = 5
14 DIV 3 = 4
13 DIV 3 = 4
12 DIV 3 = 4
11 DIV 3 = 3

Another integer operator related to integer division is the MOD operator. MOD stands for modulus, which is the remainder when you divide one integer by another.

Example 4.6 The expression 14 MOD 3 evaluates to 2, because you get a remainder of 2 when you divide 14 by 3. Further examples of MOD are:

15 MOD 3 = 0
14 MOD 3 = 2
13 MOD 3 = 1
12 MOD 3 = 0
11 MOD 3 = 2 ■

The DIV and MOD operators of Component Pascal are related by a mathematical equation. The equation is based on the following two facts.

- m div n is the quotient of $m \div n$.
- m mod n is the remainder of $m \div n$.

Let q represent the quotient and r represent the remainder, so that

$$q = m \text{ div } n$$
$$r = m \text{ mod } n$$

Then the relationship between div and mod is expressed mathematically as

$$m = q \cdot n + r \qquad 0 \le r < n$$

Example 4.7 For $m = 14$ and $n = 3$ as in Example 4.5 and Example 4.6 above, q and r are calculated as

$$q = m \text{ div } n = 14 \text{ div } 3 = 4$$
$$r = m \text{ mod } n = 14 \text{ mod } 3 = 2$$

The mathematical relationship with these numbers is

$$14 = 4 \cdot 3 + 2 \qquad 0 \le 2 < 3$$

You can see that for the given divisor $n = 3$, the remainder r will always satisfy the inequality $0 \le r < 3$. In Example 4.6, the remainders when you divide by 3 are limited to the values 0, 1, and 2, which are all less than 3. ■

The relationship between div and mod as expressed by the equation and the accompanying inequality assumes that neither the dividend m nor the divisor n are negative. If either or both of them are negative, then one or the other (or both) of the quotient q and remainder r will be negative as well. Component Pascal has a rule that describes precisely the results of the operations in that case. However, as programs in this book never use negative quotients or divisors you can safely ignore

that situation.

The procedure in Figure 4.7 uses integer expressions to compute the change in dimes, nickels, and pennies for a given number of cents with American currency. (There are 100 cents in a dollar, a dime is a 10-cent coin, a nickel is a five-cent coin, and a penny is a one-cent coin.) Integer variables are appropriate for this problem, because you cannot have a fraction of a coin. The output to the Log from procedure MakeChange is

```
You have 39 cents in change.
Dimes: 3
Nickels: 1
Pennies: 4
```

```
MODULE Pbox04E;
   IMPORT StdLog;

   PROCEDURE MakeChange*;
   VAR
      cents: INTEGER;
      dimes, nickels, pennies: INTEGER;
   BEGIN
      cents := 39;
      StdLog.String("You have "); StdLog.Int(cents);
      StdLog.String(" cents in change."); StdLog.Ln;
      dimes := cents DIV 10;
      cents := cents MOD 10;
      nickels := cents DIV 5;
      pennies := cents MOD 5;
      StdLog.String("Dimes: "); StdLog.Int(dimes); StdLog.Ln;
      StdLog.String("Nickels: "); StdLog.Int(nickels); StdLog.Ln;
      StdLog.String("Pennies: "); StdLog.Int(pennies); StdLog.Ln
   END MakeChange;

END Pbox04E.
```

Figure 4.7
The number of dimes, nickels, and pennies required for a given amount of change.

The first assignment statement computes the number of dimes by dividing the amount of change by 10 with the DIV operator. Notice that DIV does not round off the value to 4, which would be the incorrect number of dimes for the change. The second assignment statement gives cents a new value, the remainder of the change after the three dimes have been accounted for. The values for nickels and pennies are computed similarly.

Component Pascal provides two procedures for processing integers—INC and DEC, which stand for increment and decrement respectively. INC(v) adds 1 to integer variable v, INC(v, n) adds n to variable v, DEC(v) subtracts 1 from variable v, and DEC(v, n) subtracts n from variable v. Figure 4.8 summarizes the equivalent assignment statements.

You might be wondering why you would bother with these functions when it would be just as easy to use the assignment statements directly. The reason is that

Procedure	Meaning
INC(v)	v := v + 1
INC(v, n)	v := v + n
DEC(v)	v := v - 1
DEC(v, n)	v := v - n

Figure 4.8
The increment and decrement functions for integers.

the functions are designed to make use of special increment and decrement features of the computer hardware. The equivalent assignment statements may require more storage for the object program and the resulting object program may run slower than if you use the increment and decrement functions.

Mixed expressions

Component Pascal numeric expressions are similar to the mathematical expressions that you learned in algebra, but they have one important difference. Algebra usually makes no distinction between expressions for real values and expressions for integer values. However, because computers store integer values and real values with different internal codes, Component Pascal makes an important distinction between real and integer expressions.

Component Pascal permits you to use integer values in real expressions, though it does not permit you to use real values in integer expressions. This feature is another example of automatic conversion from integer to real values, as described in the discussion of Figure 4.5. When you use an integer value in a real expression, the compiler converts it to the equivalent real value before translating the expression to machine language.

Example 4.8 Suppose dollars is a real variable and cents is an integer variable. The assignment

```
dollars := dollars + cents / 100.0
```

is legal even though dollars and 100.0, which are real, are in the same expression as cents, which is integer. Because the division operator is /, not DIV, the compiler expects both operands to be real. Though the 100.0 operand is already real, the cents operand is integer, so the compiler converts it to real. Then the addition takes place between the two real operands. ▮

Example 4.9 The expression

```
dollars MOD 100
```

would be illegal if dollars is a real variable, because MOD expects its operands to be integers. There is no automatic conversion from real to integer, only from integer to real. ▮

These ideas may be a little confusing at first because the symbols for addition, subtraction, and multiplication are the same for real expressions as they are for integer expressions. (However, the symbols for division are different.) Whether an expression with +, -, or * is an integer expression or a real expression depends on its operands. If one or both of its operands is real, the result is real. If both operands are integers, the result is integer. Figure 4.9 summarizes the types of results for the arithmetic operations.

Operator	Operation	Type of operands	Type of result
+	Addition	Both integer	Integer
		At least one real	Real
-	Subtraction	Both integer	Integer
		At least one real	Real
*	Multiplication	Both integer	Integer
		At least one real	Real
/	Real division	Integers or reals	Real
DIV	Integer division	Integers	Integer
MOD	Modulus	Integers	Integer

Figure 4.9 Types of results for the arithmetic operations.

Example 4.10 Here are two examples of legal mixed expressions:

```
14.0 / (12 DIV 5)        98 / 3
14.0 / 2                 98.0 / 3.0
14.0 / 2.0               32.667
7.0
```

In each example, Component Pascal recognizes that / is a real operator and converts the operands to real values if necessary.

The ABS function

ABS(x) is a Component Pascal function that returns the absolute value of x. It is unusual because the type that it returns depends on the type of the parameter x. If the type of x is integer the type of the returned value is integer, and if the type of x is real the type of the returned value is real.

Example 4.11 The function ABS(-3) returns integer 3, ABS(3) returns integer 3, and ABS(-3.7) returns real 3.7.

In mathematics, there is no largest integer. There is no upper limit on the value that an integer variable can have. But all computers have finite storage capacity for storing numeric values. Fortunately, Component Pascal allows you to store fairly large values in your numeric variables. An value of type INTEGER can store values

in the range

–2,147,483,648 .. 2,147,483,647

If you ever need to store values larger than two billion you have the option of declaring a variable of type LONGINT, which can store values in the range

–9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

You can assign an integer expression to a long integer variable. Component Pascal will provide automatic conversion from integer to long integer similar to how it provides automatic conversion from integer to real. But you cannot assign a long integer expression to an integer variable.

Example 4.12 If myInt is an integer variable, and myLongInt is a long integer variable, then

```
myInt := myLongInt
```

is not legal, but

```
myLongInt := myInt
```

is legal. ■

The SHORT function

If you ever have a long integer value that you need to assign to an integer variable you can use the SHORT(x) function to do the conversion. If the type of x is long integer, then SHORT returns the equivalent integer.

Example 4.13 The following expression is legal.

```
myInt := SHORT(myLongInt)
```

■

The ENTIER function

If you have a real value that you need to use in an integer expression, Component Pascal provides a function called ENTIER(x). It takes a real value for x and returns a long integer value as the truncated value of x.

Example 4.14 If dollars is a real variable and bigBills is a long integer variable, then

```
bigBills := ENTIER(dollars)
```

truncates the value of dollars, converts it to a long integer, and assigns it to bigBills. If dollars has the value 4.95, then bigBills gets 4. You must write

```
bills := SHORT(ENTIER(dollars))
```

if bills is an integer variable. ■

If you ever need to use the maximum value of an integer, you do not need to remember the 10-digit sequence. MAX is a built-in function that takes a type for the actual parameter and returns the maximum value for that type. Similarly, the MIN function returns the minimum value for a type.

The MAX and MIN functions for types

Example 4.15 The statements

```
StdLog.String("MAX(INTEGER) = "); StdLog.Int(MAX(INTEGER)); StdLog.Ln;
StdLog.String("MIN(INTEGER) = "); StdLog.Int(MIN(INTEGER)); StdLog.Ln;
```

produce the following output on the Log:

```
MAX(INTEGER) = 2147483647
MIN(INTEGER) = -2147483648
```

MAX and MIN can be used with types other than integer. If you output the maximum value of the real type, you will discover that it is about 1.798×10^{308}.

MAX and MIN are unusual on two counts. First, most functions take variables or constants for their actual parameters, while MAX and MIN take a type. Second, there is another form of MAX and MIN that does take variables and constants. If you supply the MAX function with two actual parameters, it will return the larger of the two. Similarly, MIN will return the minimum of two actual parameters.

The MAX and MIN functions for variables and constants

Example 4.16 If myData is an integer variable, the statements

```
myData := 7;
StdLog.String("The larger is"); StdLog.Int(MAX(myData, 5)); StdLog.Ln;
```

produce the following output on the Log:

```
The larger is 7
```

because 7 is greater than 5. In this example, MAX has variable myData for the first actual parameter and constant 5 for the second.

Function procedures

BlackBox provides module Math, a standard library that is documented on-line. A few of the many functions from the interface are listed in Figure 4.10. Math.Pi() always returns the value of π. Math.Exp(x) raises the base of the natural logarithms, e, to the power specified by the parameter x. Math.Ln(x) returns the natural logarithm of x, and Math.Log(x) returns the base-10 logarithm. The angles of the trigonometric functions are always expressed in radians, not degrees.

```
DEFINITION Math;

   PROCEDURE Pi (): REAL;

   PROCEDURE Sqrt (x: REAL): REAL;
   PROCEDURE Exp (x: REAL): REAL;
   PROCEDURE Ln (x: REAL): REAL;
   PROCEDURE Log (x: REAL): REAL;
   PROCEDURE Power (x, y: REAL): REAL;
   PROCEDURE IntPower (x: REAL; n: INTEGER): REAL;

   PROCEDURE Sin (x: REAL): REAL;
   PROCEDURE Cos (x: REAL): REAL;
   PROCEDURE Tan (x: REAL): REAL;
   PROCEDURE ArcSin (x: REAL): REAL;
   PROCEDURE ArcCos (x: REAL): REAL;
   PROCEDURE ArcTan (x: REAL): REAL;

END Math.
```

Figure 4.10
Some of the math functions from the interface of the Math module.

Proper procedures and function procedures

Component Pascal provides two types of procedures—proper procedures and function procedures. The procedures in module StdLog shown in its interface in Figure 3.4 are all proper procedures. The procedures listed in the interface for module Math are all function procedures. You can tell from an interface whether a procedure is a function procedure by inspection of its formal parameters. If the formal parameters include a colon : followed by a type to the right of the parentheses ( ), the procedure is a function procedure. Otherwise it is a proper procedure.

Function procedures are similar to functions in mathematics, where $f(x)$ usually means a function of x. If you supply a value for x, the function will return a value for $f(x)$. In the specification of a function procedure, the type following the parentheses is the type of the value returned by the function procedure.

Example 4.17 The interface for function procedure IntPower

```
PROCEDURE IntPower (x: REAL; n: INTEGER): REAL
```

specifies that the first parameter must be compatible with real, the second parameter must be compatible with integer, and the value returned by the function will have type real. If alpha has type real, then the assignment

```
alpha := Math.IntPower(2.4, 3)
```

is legal. The function procedure returns the real value 13.824, which is then assigned to alpha. The assignment would not be legal if alpha had type integer because you cannot assign a real value to an integer variable. ∎

In the same way that you can assign an integer value to a real variable because of the automatic conversion from integer to real, you can supply an integer actual

parameter to a real formal parameter. But you cannot supply a real actual parameter to an integer formal parameter.

Example 4.18 The assignment statement

```
alpha := Math.IntPower(2, 3)
```

is legal even though 2 is an integer and x is a real. However, the assignment statement

```
alpha := Math.IntPower(2, 3.0)
```

is not legal because 3.0 is a real but n is an integer. ■

Loosely speaking, an arithmetic expression is a combination of real values, integer values, variable identifiers, operators, functions, and parentheses. The exact syntax is specified in Appendix A. However, your experience from mathematics is probably sufficient to recognize an illegal expression.

Example 4.19 The following examples are valid expressions, assuming that a and b are real variables, and i and j are integer variables.

```
a * (b + 4.7)          2 * (3 + 4 * (i + 1))
2.1                    -3.4 * Math.Sin(ABS(b))
j                      Math.Cos(Math.Pi() / 4.0)
```
■

Example 4.20 An example of an illegal expression is

```
a * ((b + 4.7)
```

because one of the left parentheses does not have a matching right parenthesis. ■

Character variables

Component Pascal has several types that are not numeric, one of which is CHAR. CHAR stands for character. A variable that has type CHAR can have a value that is a single letter or punctuation mark or digit, not limited to the English alphabet. The possible values include characters from most of the languages in the world, as specified by the Unicode character standard. The character values are the ones that are printed on the keycaps of the keyboard. Example of character values are: R, r, E, e, $, and 4. In a Component Pascal program listing, character values are enclosed in single quote or double quote marks.

Example 4.21 You could declare the following variables in a procedure

```
char1, char2, char3: CHAR;
```

A valid sequence of assignment statements would then be

```
char1 := 'b'; char2 := 'u'; char3 := 't'
```

The following output statements

```
StdLog.Char(char3); StdLog.Char(char2); StdLog.Char(char1)
```

would then produce the output

```
tub
```

on the Log.

The decimal digits are included in the Unicode character set. There is a difference between the character '4' and the integer 4.

Example 4.22 In the previous example, the assignment statement

```
char1 := '4'
```

would be legal, but the assignment statement char1 := 4 would not, because char1 has type CHAR and 4 has type INTEGER.

It is occasionally useful to process characters with the arithmetic operators. Because the arithmetic operators cannot operate on characters directly, Component Pascal provides a means for transforming between characters and integers. To perform an arithmetic operation on a character, you first convert it to an integer, then perform the operation on the integer, then convert the integer back to a character. The transformation between characters and integers is based on the fact that each character has a place on the integer number line. Figure 4.11 shows the characters below the number line with their associated integer values above the line.

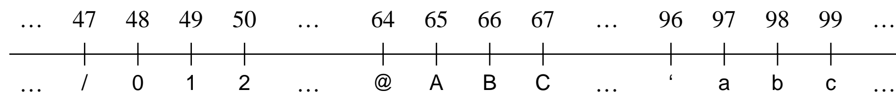


Figure 4.11 The number line for some of the character values.

The integer above a character is called its ordinal value. For example, the ordinal value of the character 'B' is 66, and the ordinal value of the character '1' is 49. The two functions that Component Pascal provides for converting between characters and integers are ORD and CHR. ORD takes a character for its actual parameter and returns its corresponding ordinal value. CHR takes an integer for its actual parameter and returns the corresponding character.

Example 4.23 Suppose myCh is a variable that has type character and myInt is a variable that has type integer. You want to change myCh to have the value of the next letter in the alphabet. This operation corresponds to adding 1 to the character, but

Component Pascal does not permit addition on characters. The following statements perform the conversion using ORD and CHR:

```
myInt := ORD(myCh);
INC(myInt);
myCh := CHR(myInt)
```

A more economical way to do the same thing is to dispense with the integer variable altogether and write the single statement

```
myCh := CHR(ORD(myCh) + 1)
```

In either case, if myCh has the value S before execution it will have the value T after execution. ∎

The PboxStrings module

The BlackBox framework provides a module called Strings that has several procedures for operating on characters. Some of the procedures in Strings, however, are difficult for beginning programmers to use. Consequently, this author has written a module called PboxStrings that contains procedures similar to those contained in Strings. The procedures are easier to use than those in Strings and are designed as an aid to presenting the material in this book. The PboxStrings module is contained in the Pbox project folder, which is not part of the standard BlackBox distribution. Your instructor can give you a copy of the Pbox modules, or you can obtain them from the author over the Internet. The URL for the World Wide Web site is

`ftp://ftp.pepperdine.edu/pub/compsci/prog-bbox/`

The URL for the Pbox project

Note that this URL begins with `ftp://` and not the usual `http://`. The site contains not only the Pbox project folder, but also the source code for every program in this book. You should be aware that any software you develop with the Pbox modules will not be usable on a computer that does not have the Pbox project installed.

Figure 4.12 is the interface for PboxStrings. It includes function Lower, which converts a character to lowercase, and Upper, which converts to uppercase.

```
DEFINITION PboxStrings;

    PROCEDURE Lower (ch: CHAR): CHAR;
    PROCEDURE Upper (ch: CHAR): CHAR;
    PROCEDURE ToLower (from: ARRAY OF CHAR; OUT to: ARRAY OF CHAR);
    PROCEDURE ToUpper (from: ARRAY OF CHAR; OUT to: ARRAY OF CHAR);
    PROCEDURE IntToString (n, minWidth: INTEGER; OUT s: ARRAY OF CHAR);
    PROCEDURE RealToString (x: REAL; minWidth, dec: INTEGER; OUT s: ARRAY OF CHAR);

END PboxStrings.
```

Figure 4.12
The interface for PboxStrings.

Example 4.24 If myCh is a variable that has type character and value 'B', then the statement

```
myCh := PboxStrings.Lower(myCh)
```

changes its value to 'b'. If the same variable has a lowercase value, say 'h', before execution of the statement, its value will not be changed when the statement executes. ▮

Character arrays

Characters are more useful when you string them together to form words and sentences. In Component Pascal, you can string values together with a construction called an *array*. An array is simply a collection of values, all of which must have the same type. A character array can have a value that is a string. Figure 4.13 is an example of a procedure that declares variable message to have type character array. Procedure PrintString prints the text What's up, Doc? to the Log.

An array is a collection of values, all with the same type.

```
MODULE Pbox04F;
   IMPORT StdLog;

   PROCEDURE PrintString*;
   VAR
      message: ARRAY 128 OF CHAR;
   BEGIN
      message := "What's up, Doc?";
      StdLog.String(message); StdLog.Ln
   END PrintString;

END Pbox04F.
```

Figure 4.13
A procedure that declares a variable with string type.

The individual characters that form a string are stored consecutively in the memory of the computer. A special character, written 0X in Component Pascal, is also stored after the last character to serve as a marker for the end of the string. When procedure PrintString declares the variable message to be an array of 128 characters, it is declaring that the string value of message can have as many as 127 characters, because one spot in the array must contain the last 0X character. Figure 4.14 shows how the characters in variable message are stored.

W	h	a	t	'	s		u	p	,		D	o	c	?	0X			…

Figure 4.14
Storage of a string value in an array of characters.

When you declare a variable to be an array of characters you must decide how many characters to allocate. The size of the array should be a bit larger than the

longest string you would expect to store in the variable. If you make all your arrays excessively large you will be wasting memory. For example, if the array is to store the last name of a person, 128 characters would be way too many. Perhaps 32 would be more reasonable, because few people have last names with more than 31 characters.

The procedure StdLog.String can take as its actual parameter a variable of type character array as well as a string. In Figure 3.6 the actual parameter of StdLog.String is the string "Mr. K. Kong", but in Figure 4.13 the actual parameter of StdLog.String is the variable message that has type character array.

The + symbol for concatenation

Chapter 2 introduced the concatenation operation on strings of letters. Component Pascal uses the + symbol for concatenation when it is placed between strings or character arrays. Figure 4.15 shows a procedure whose output is identical to that of the procedure in Figure 4.4. It uses the + symbol to concatenate several strings.

```
MODULE Pbox04G;
   IMPORT StdLog, PboxStrings;

   PROCEDURE Change*;
   VAR
      cents: INTEGER;
      centString: ARRAY 16 OF CHAR;
      message: ARRAY 64 OF CHAR;
   BEGIN
      cents := 39;
      PboxStrings.IntToString(cents, 1, centString);
      message := "You have " + centString + " cents in change.";
      StdLog.String(message); StdLog.Ln
   END Change;

END Pbox04G.
```

Figure 4.15
A procedure that uses the + operator to concatenate strings. It imports the PboxStrings module.

The meaning of OUT in a formal parameter list

Figure 4.12 shows the interface for PboxStrings. Proper procedure IntToString has three formal parameters—n, minWidth, and s. Notice that s is preceded by the reserved word OUT. A formal parameter preceded by OUT is designed to change the value of its actual parameter. In this program, the actual parameter is centString. Procedure IntToString will change the value of centString when it executes.

Here is how the program works. The variable declaration in procedure Change of Figure 4.15 declares cents to have type INTEGER. During execution, the first assignment statement gives the value 39 to cents. Then the IntToString statement makes a string image of the value. The second parameter in an IntToString call specifies the minimum field width. This IntToString call specifies a minimum field width of one because the value displayed on the Log will appear in the middle of a sentence. Because the field width will expand, if necessary, to fit all the digits into the display, this technique guarantees proper spacing within the sentence. Figure 4.16 shows the value of centString after the call to IntToString is completed.

The next statement concatenates the string "You have " with the value of centString, then concatenates that with the string " cents in change.", and assigns the result to the character array message. Finally, StdLog.String prints the value of mes-

3	9	0X													

Figure 4.16
The value of centString.

sage to the Log.

Suppose you specify a field width of 2, anticipating that the value of the variable will require exactly two digits to display. If the value is 39, as in Figure 4.15, the output will be unchanged. But if the value is 8 instead of 39 and you still specify a field width of 2, the output would be

```
You have  8 cents in change.
```

with an extra space before the 8. If you specify a field width of 1, the spacing will always be correct in the sentence regardless of how many digits are required to display the value.

Procedure RealToString works like IntToString except that it has four parameters instead of three—x, minWidth, dec, and s. x is the real value for which you want the string display. minWidth and s are the minimum field width and the resulting string as with procedure IntToString. dec allows you to specify how many places past the decimal point you want to include. RealToString rounds off fractional values as you would expect.

Example 4.25 Suppose amtOwed is a variable that has type real and value 84.376. It represents a dollar amount, and you want to display the value to the nearest cent, which is two places past the decimal point. Assuming that message and dollarString are arrays of characters, the following statements

```
PboxStrings.RealToString(amtOwed, 1, 2, dollarString);
message := "You owe " + dollarString + " dollars.";
StdLog.String(message); StdLog.Ln
```

will produce

```
You owe 84.38 dollars.
```

on the Log. ■

It is sometimes necessary to assign one character array that has a string value to another character array. Component Pascal executes the assignment by copying every value in the array regardless of the number of characters in the string.

Example 4.26 Suppose myString and yourString are both declared as follows.

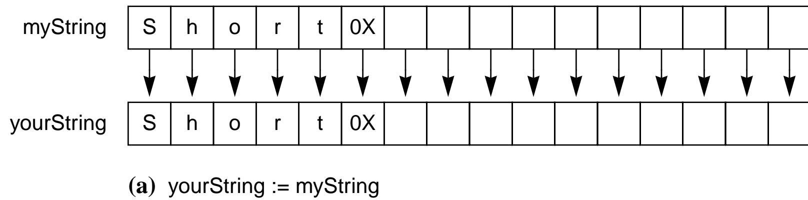
```
VAR
   myString, yourString: ARRAY 16 OF CHAR;
```

If myString has previously been given the value "Short", then the assignment state-

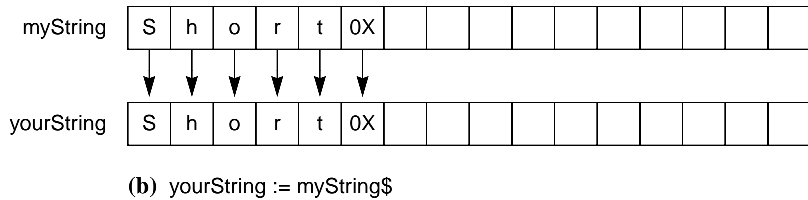
ment

```
yourString := myString
```

makes 16 copies as shown in Figure 4.17(a), even though the string has only five characters. ▮



(a) yourString := myString



(b) yourString := myString$

Figure 4.17
Character array assignments.

The $ selector

The problem of unnecessary character copies occurs because you must allocate more memory than is required by most of the string values stored in the array. Component Pascal provides a $ selector that you can use to eliminate the unnecessary character copies during array assignment. Appending the $ selector to the name of a character array changes its designation to include only the characters from the first position up to and including the 0X.

Example 4.27 With myString and yourString declared as in Example 4.26, the name myString$ designates the five characters "Short" plus the 0X character. The assignment

```
yourString := myString$
```

makes only six copies as shown in Figure 4.17(b). ▮

★ Guarded command language

The starred sections of this book are for those who have studied, or are currently studying, formal methods. Formal methods are the mathematical foundation of most programming languages, including Component Pascal and are increasingly important in the field of software engineering. If you have not yet learned formal methods you may omit these sections.

The purpose of GCL

The goal of the starred sections is to show the application of formal methods to computing practice with Component Pascal. The goal is not to teach principles of

formal methods, which is outside the scope of this book. A common language used to analyze algorithms with formal methods is the guarded command language, which this book abbreviates as GCL. This section introduces GCL and shows the relationship between GCL and Component Pascal (CP).

One difference between GCL and CP are the goals and intended use of each language. The goal of GCL is to provide a convenient mathematical notation for proving the correctness of programs. It is typically used by hand with pencil and paper. The notation is, therefore, short and succinct to minimize the amount of handwriting. Variable names are purposely kept short, typically only one letter long. Such a practice is preferred in GCL but discouraged in CP where the services of a document editor permit longer, more descriptive names to be used with ease to enhance readability.

GCL does not require a separate VAR section to declare the type of a variable. Instead, you simply write the variable followed by a colon, followed by the symbol for its type from Figure 4.18.

CP	GCL
INTEGER	$\mathbb{Z}$
REAL	$\mathbb{R}$

Figure 4.18
Specifying type in GCL.

Example 4.28 A CP program that contains the variable section

```
VAR
    cents: INTEGER;
    dollarAmount: REAL;
```

would be written in GCL as

c: $\mathbb{Z}$
d: $\mathbb{R}$

where the single variable name c is used in place of the longer, more descriptive name cents and similarly with d for dollarAmount. ∎

Assignment in GCL

Fortunately, the assignment statement := is the same in both CP and GCL. The state of a computation is a list of the variables and their values. The effect of an assignment statement is to change the state of the computation by changing the values.

Example 4.29 With the variables declared as in Example 4.28, suppose the state of the computation is (c, 47), (d, 89.60). The assignment statement to add 5% to dollarAmount in CP

```
dollarAmount := dollarAmount * 1.05
```

is written

$d := d * 1.05$

in GCL and changes the state to (c, 47), (d, 94.08). ∎

In GCL, you can combine the type information for a variable with any expres-

sion. This technique saves a little extra writing.

Example 4.30 The type declaration of Example 4.28 could be combined with the assignment statement of Example 4.29 as

d: ℝ := d * 1.05

Alternatively, the type information could be combined with the initial state as

(c: ℤ, 47), (d: ℝ, 89.60) ∎

As in CP, the semicolon in GCL represents a sequence of statements. That is, the statements are executed in order not simultaneously.

Example 4.31 The assignment statements from Figure 4.7 are written in GCL as

d := c div 10; c := c mod 10; n := c div 5; p := c mod 5 ∎

Multiple assignment

In addition to sequencing statements with the semicolon symbol, you can perform multiple assignments in GCL, a feature that is not available in CP. With multiple assignments, the values are changed simultaneously. To translate between multiple assignments in GCL and sequential assignments in CP you can sometimes simply make the multiple assignments sequential and the computations will be equivalent. However, if the first assignment in a sequence changes the value of a variable that is in turn used in an expression on the right side of a later assignment, the translation will be incorrect. In such a case, you will need to resort to a temporary variable in the sequential version.

Example 4.32 The multiple assignment in GCL

c, n := c mod 10, c div 5

is not equivalent to the CP sequence

```
cents := cents MOD 10;
nickels := cents DIV 5
```

If the initial state is (c, 39), (n, ?), then the final state after the multiple assignment will be (c, 9), (n, 7), because 39 div 5 is 7. However, the final state after the sequential assignment will be (c, 9), (n, 1), because 9 div 5 is 1. On the other hand, the multiple assignment

d, c := c div 10, c mod 10

is equivalent to

```
dimes := cents DIV 10;
cents := cents MOD 10
```

∎

Example 4.33 If you want to exchange the values of x and y in GCL you can simply write the multiple assignment

$x, y := y, x$

which, for example, would change the state $(x, 3), (y, 14)$ to $(x, 14), (y, 3)$. However, the corresponding sequence in CP

```
x := y; y := x
```

would change the state $(x, 3), (y, 14)$ to $(x, 14), (y, 14)$, because the modified value of x is assigned to y instead of the original value of x. To exchange the values requires a temporary variable, say t, to store the original value of x so that it can be assigned to y.

```
t := x; x := y; y := t
```

■

Exercises

1. Inspect the interface of module TextViews on-line in BlackBox using the technique of Figure 3.3, and answer the following questions about the procedures that are listed in it.

(a) How many modules are listed in the IMPORT list of TextViews?
(b) State whether each of the following is a proper procedure or a function procedure: Deposit, Focus, ShowRange, ThisRuler.
(c) How many parameters does ShowRange have? What are their names?
(d) How many parameters does ThisRuler have? What is the type of its returned value?

2. Evaluate the following expressions. Indicate real results in your answer with a decimal point and integer results by not including a decimal point. If the expression is illegal, explain why.

(a) 5.0 / 2.0
(b) 5 / 2
(c) 5 DIV 2
(d) 5.0 DIV 2.0
(e) 5 MOD 2
(f) 5.0 MOD 2
(g) ENTIER(8.3)
(h) ENTIER(8.7)
(i) ABS(-6.8)
(j) ABS(6)
(k) Math.IntPower(3.0, 2)
(l) Math.IntPower(3.0, 2.0)
(m) Math.Sqrt(16.0)
(n) Math.Sqrt(16)
(o) Math.Sin(0.0)
(p) Math.Exp(1.0)
(q) Math.Ln(Math.Exp(4.7))

3. Evaluate the following expressions. Indicate real results in your answer with a decimal point and integer results by not including a decimal point. If the expression is illegal, explain why.

(a) 7.0 / 3.0 **(b)** 7 / 3 **(c)** 7 DIV 3
(d) 7.0 DIV 3.0 **(e)** 7 MOD 3 **(f)** 7.0 MOD 3
(g) ENTIER(7.3) **(h)** ENTIER(7.9) **(i)** ABS(-4.8)
(j) ABS(4) **(k)** Math.IntPower(4.0, 2) **(l)** Math.IntPower(4.0, 2.0)
(m) Math.Sqrt(9.0) **(n)** Math.Sqrt(9) **(o)** Math.Sin(0.0)
(p) Math.Exp(1.0) **(q)** Math.Ln(Math.Exp(5.1))

4. Evaluate the following expressions. Indicate a character result in your answer by enclosing it in quotes. If the expression is illegal, explain why.

(a) ORD('b') **(b)** CHR(50) **(c)** @ + 1
(d) '@' + 1 **(e)** ORD('@' + 1) **(f)** ORD(@) + 1
(g) ORD('@') + 1 **(h)** CHR(ORD('@') + 1) **(i)** 'a' - 'A'
(j) CHR(ORD('D') + ORD('a') - ORD('A'))

5. Evaluate the following expressions. Indicate a character result in your answer by enclosing it in quotes. If the expression is illegal, explain why.

(a) ORD('B') **(b)** CHR(49) **(c)** / + 1
(d) '/' + 1 **(e)** ORD('/' + 1) **(f)** ORD(/) + 1
(g) ORD('/') + 1 **(h)** CHR(ORD('/') + 1) **(i)** 'a' - 'A'
(j) CHR(ORD('E') + ORD('a') - ORD('A'))

6. i and j are integer variables, and x is a real variable. Determine the values of each of the variables after the sequence of assignments statements executes. Indicate real values with a decimal point and integer values by not including a decimal point.

(a)
```
i := 18;
j := i DIV 7;
x := 4.5;
INC(i);
x := x + i * 2
```

(b)
```
j := 14;
i := j MOD 5;
x := 2.7;
INC(j);
x := x + j * 2
```

(c)
```
i := 3;
j := 18;
x := 7.9;
i := j;
j := i
```

7. Write the mathematical relation between DIV and MOD, including the inequality, for dividend 27 and divisor 6.

8. In Example 4.17, **(a)** is x a formal parameter or is it an actual parameter? **(b)** is 2.4 a formal parameter or is it an actual parameter?

9. If myMessage and yourMessage are both declared to be ARRAY 128 OF CHAR, and myMessage has the string value "Look out!", **(a)** how many characters are copied with the assignment yourMessage := myMessage? **(b)** How many with the assignment yourMessage := myMessage$?

10. Write the equivalent CP program statements for the following GCL statements. For each part, write the final state if the initial state is $(x, 1), (y, 2), (z, 3)$.

(a) $x, y := x + z, y * x$
(b) $x, y, z := x + z, y + x, z + y$
(c) $x, y, z := y + 4, y + x, z + y$

Problems

11. Write a procedure with integer variable feet and real variables inches and meters. Assign feet and inches values and compute the equivalent length in meters. One inch is exactly 0.0254 meters and one foot is exactly 12 inches. Use StdLog.Int and StdLog.Real to output the values identified appropriately. Here is a sample output to the Log.

```
Feet: 4
Inches: 3.8
Meters: 1.31572
```

12. Work Problem 11, but display the computed value for meters to two places past the decimal. Use StdLog.Int and StdLog.Real for feet and inches and PboxStrings.RealToString with a character array variable for meters.

13. Write a procedure with two real variables for the temperature in Fahrenheit and Celsius. Assign a value to the variable for Fahrenheit and compute the equivalent temperature in Celsius. Show both temperatures on the Log with their values identified appropriately as the values are in Problem 11.

14. Work Problem 13, but display the computed value for Celsius to one place past the decimal. Use StdLog.Real for the Fahrenheit value and PboxStrings.RealToString with a string variable for the Celsius value.

15. Write a procedure with two real variables for the lengths of two perpendicular sides of a right triangle and a third variable for the length of the hypotenuse. Assign values to the variables for the sides and compute the value for the hypotenuse. Show all three lengths on the Log with their values identified appropriately as the values are in Problem 11.

16. Work Problem 15, but display the computed value for the hypotenuse to one place past the decimal. Use StdLog.Real for the side values and PboxStrings.RealToString with a string variable for the hypotenuse value.

17. Write a procedure with a real variable for the radius of a circle and two additional real variables for its circumference and area. Assign values to the variables for the radius and compute the values for the circumference. Import the value of π from module Math for your computations. Show all three measures on the Log with their values identified appropriately as the values are in Problem 11.

18. Work Problem 17, but display the computed values for the circumference and area to one place past the decimal. Use StdLog.Real for the radius value and PboxStrings.RealToString with a string variable for the circumference and radius values.

19. Modify the program in Figure 4.7 to make change for quarters as well as dimes, nickels, and pennies. (A quarter is a 25-cent coin.)

20. Write a procedure with three integer variables for the number of hours, days, and weeks. Assign a value to the variable for hours and compute the equivalent number of days, weeks, and hours. Show all time measures on the Log with their values identified

appropriately as the values are in Problem 11. For example, if the variable for hours is assigned 4123, the output should be

```
Total hours: 4123
Number of weeks: 24
Number of days: 3
Number of hours: 19
```