However, for most programs the most convenient way to handle the array is to use a two-dimensional PACKED ARRAY OF BOOLEAN as described previously.

TEXT AS GRAPHICS: WCHAR, WSTRING, AND CHARTYPE

Three procedures allow you to put characters on the graphics screen. If the turtle is at (X,Y) you can use these procedures to put a character or string on the screen with its lower left corner at (X,Y). Each character occupies a rectangular area 7 dots wide and 8 dots high on the screen.

These procedures use an array stored in the file SYSTEM.CHARSET on diskette APPLE1:. This array contains all the characters used, and is read in by the initialization routine when your program USES TURTLEGRAPHICS. If you make up an array containing your own character set, you should rename the old SYSTEM.CHARSET and then name your new array SYSTEM.CHARSET (see note at the end of this chapter).

WSTRING and WCHAR use the procedure DRAWBLOCK to copy each character from the array onto the screen. The MODE parameter that they use is set by the CHARTYPE procedure.

The WCHAR procedure has the form

```
     WCHAR (CH)
```

where CH is a an expression of type CHAR. This procedure places the character on the screen with its lower left corner at the current location of the turtle. When this procedure is used, the turtle is shifted to the right 7 dots from its old position. For example, this puts an X in the center of the screen:

```
     PENCOLOR (NONE);
     MOVETO (137,9Ø);
     WCHAR ('X')
```

In this example, note that it was not necessary to specify a new pen color before calling WCHAR. The character is not plotted with the current pen color; rather it depends on the current MODE, just as DRAWBLOCK does. For details, see CHARTYPE below.

The CHAR value passed to WCHAR is restricted to the first 128 characters of the ASCII set as shown in Table 7 of Appendix B.

The WSTRING procedure has the form

```
     WSTRING (S)
```

where S is an expression of type STRING. An entire string of characters is placed on the screen with the lower left corner of the first character at the current turtle position. The turtle is shifted 7 dots to the right for each character in the string. This procedure calls WCHAR for each character in the string.

The characters in the STRING value passed to WSTRING are restricted to the first 128 characters of the ASCII set as shown in Table 7 of Appendix B.

The CHARTYPE procedure has the form

```
     CHARTYPE (MODE)
```

where MODE is an integer selecting one of the 16 MODEs described above for DRAWBLOCK. MODE defines the way characters get written on the screen; it works for WCHAR and WSTRING just as it works for DRAWBLOCK.

The default MODE is 1Ø, which places each character on the screen in white, surrounded by a black rectangle. MODE 5 is the inverse of MODE 1Ø: each character is in black surrounded by a white rectangle.

One of the most useful other MODEs is 6, which does an exclusive OR of the character with the current contents of the screen. If you use MODE 6 to draw a character or string and then redraw it at the same location with MODE 6, the effect is to erase the character or string, leaving the original image unaffected. This is especially useful for user messages in a graphics-oriented program.

If you wish to create your own character set file for use with WCHAR and WSTRING, it must be structured as follows:

- The file consists of 1Ø24 bytes.
- Starting with the first byte in the file, each character in the character set is represented by 8 contiguous bytes.
- Each byte represents one row of 8 dots in the character image. The first byte of each character representation is the bottom row of the image.

- The least significant bit of each byte is the leftmost dot in the row.

- The most significant bit of each byte is ignored; the rows are only seven dots each.

Such a file can be created either in assembly language or in Pascal. In Pascal, you can build the character representations in memory as packed arrays of the type Ø..255 since each element of such an array is in effect a byte. For example, you might use the declarations

```
TYPE CHARIMAGE=PACKED ARRAY[Ø..7] OF Ø..255;
     CHARSET=PACKED ARRAY[Ø..127] OF CHARIMAGE;
     CHARFILE=FILE OF CHARSET;

VAR CHARACTERS:CHARSET;
    OUTFILE:CHARFILE;
```

OTHER SPECIAL APPLE FEATURES: THE APPLESTUFF UNIT

This section tells you how to generate random numbers, how to use the game paddle and button inputs, how to read the cassette audio input, how to switch the game-control's TTL outputs and how to generate sounds on the Apple's speaker. To use these special Apple features from Pascal, you first have to place the declaration

```
USES APPLESTUFF;
```

immediately after the program heading. If you wish to use both turtle graphics and the Apple features you would say

```
USES TURTLEGRAPHICS, APPLESTUFF;
```

since there can only be one USES in a program.

THE RANDOM FUNCTION

RANDOM is an integer function with no parameters. It returns a value from Ø through 32767. If RANDOM is called repeatedly, the result is a psuedo-random sequence of integers. The statement

```
WRITELN (RANDOM)
```

will display an integer between the indicated limits.

A typical application of this function is to get a pseudo-random number, say, between LOW and HIGH inclusive. The expression

```
LOW + RANDOM MOD (HIGH-LOW+1)
```

is sometimes used where results are not critical, but the values formed by this expression are <u>not</u> evenly distributed over the range LOW through HIGH. If you want pseudo-random integers evenly distributed over a range, you can use the following function:

```
  FUNCTION RAND (LOW, HIGH:INTEGER; VAR ERROR:BOOLEAN):INTEGER;
    VAR MX, C, D: INTEGER;
    BEGIN
      RAND := Ø;
      ERROR := TRUE;
      IF LOW > HIGH THEN EXIT(RAND); (*error exit*)
      IF LOW <= Ø THEN
        IF HIGH > MAXINT + LOW THEN EXIT(RAND); (*error exit*)

      ERROR := FALSE; (*no errors*)
      IF LOW = HIGH THEN RAND := LOW
                    ELSE BEGIN
                           C := HIGH - LOW + 1;
                           MX := (MAXINT - HIGH + LOW) DIV C + 1;
                           MX := MX * (HIGH - LOW) + (MX - 1);
                           REPEAT D := RANDOM UNTIL D <= MX;
                           RAND := LOW + D MOD C
                         END
    END;
```

If HIGH is not greater than LOW, or the difference between HIGH and LOW would exceed MAXINT, RAND returns Ø and sets the ERROR parameter to true. Otherwise, RAND returns evenly distributed pseudo-random integer values between LOW and HIGH (inclusive).

THE RANDOMIZE PROCEDURE

RANDOMIZE is a procedure with no parameters. Each time you run a given program using RANDOM, you will get the same random sequence unless you use RANDOMIZE.

RANDOMIZE uses a time-dependent location to generate a starting point for the random number generator. The starting point changes each time you do any input or output operation in your program. If you use no I/O, the starting point for the random sequence does not change.

THE KEYPRESS FUNCTION

This function, which has no parameters, returns true if a key has been pressed on the keyboard since the program started or since the last time the keyboard was read (whichever is most recent). KEYPRESS does not read the character from CONSOLE or KEYBOARD or have any other effect on I/O. The statement

```
    IF KEYPRESS THEN READ(KEYBOARD, CH)
```

(where CH is a CHAR variable) has the effect of reading the last character typed on the keyboard. This could be used to retrieve a character typed while the program was doing something else -- for instance, displaying graphics.

Once KEYPRESS becomes true it remains true until a GET, READ, or READLN accesses either the INPUT file or the KEYBOARD file, or until a UNITREAD accesses the keyboard device.

KEYPRESS does not work with an external terminal connected via a serial interface card. It will always return FALSE with such a terminal.

PADDLE, BUTTON, AND TTLOUT

The PADDLE function has the form

```
    PADDLE (SELECT)
```

where SELECT is an integer treated modulo 4 to select one of the four paddle inputs numbered Ø, 1, 2, and 3. PADDLE returns an integer in the range Ø to 255 which represents the position of the selected paddle. A 15ØK variable resistance can be connected in place of any of the four paddles.

If you try to read two paddles too quickly in succession, e.g.

```
    WRITELN (PADDLE (Ø));
    WRITELN (PADDLE (1))
```

the hardware will not be able to keep up. A suitable delay is given by the loop

```
    FOR I := Ø TO 3 DO;
```

The BUTTON function has the form

```
    BUTTON (SELECT)
```

where SELECT is an integer treated modulo 4 to select one of the three button inputs numbered Ø, 1, and 2, or the audio cassette input numbered 3. The BUTTON function returns a BOOLEAN value of TRUE if the selected game-control button is pressed, and FALSE otherwise.

When BUTTON(3) is used to read the audio cassette input, it samples the cassette input which changes from TRUE to FALSE and vice versa at each zero crossing of the input signal.

There are four TTL level outputs available on the game connector along with the button and paddle inputs. The TTLOUT procedure is used to turn these outputs on or off. TTLOUT has the form

```
     TTLOUT (SELECT, DATA)
```

where SELECT is an integer treated modulo 4 to select one of the four TTL outputs numbered Ø, 1, 2, and 3. DATA is a BOOLEAN expression.

If DATA is TRUE, then the selected output is turned on. It remains on until TTLOUT is invoked with the DATA set to FALSE.

MAKING MUSIC: THE NOTE PROCEDURE

The NOTE procedure has the form

```
     NOTE (PITCH, DURATION)
```

where PITCH is an integer from Ø through 5Ø and DURATION is an integer from Ø through 255.

A PITCH of Ø is used for a rest, and 2 through 48 yield a tempered (approximately) chromatic scale. DURATION is in arbitrary units of time.

NOTE (1,1) gives a click.

A chromatic scale is played by the following program:

```
     PROGRAM SCALE;

     USES APPLESTUFF;
     VAR PITCH, DURATION: INTEGER;

       BEGIN

         DURATION := 1ØØ;
         FOR PITCH := 12 TO 24 DO
           NOTE (PITCH, DURATION)

       END.
```

TRANSCENDENTAL FUNCTIONS: THE TRANSCEND UNIT

In Apple Pascal, the transcendental functions are not built into the language. To use this set of functions you must place the declaration

```
     USES TRANSCEND;
```

immediately after the PROGRAM heading. If you wish to use, say, APPLESTUFF with the transcendental functions, you would write

```
     USES TRANSCEND, APPLESTUFF;
```

All ANGLE and NUMBER arguments are real, and the ANGLE arguments are in radians. All of these functions return real values, and values returned by the ATAN function are in radians. The following functions are provided:

```
     SIN (ANGLE)

     COS (ANGLE)

     EXP (NUMBER)

     ATAN (NUMBER)          (Note: this is the same function
                              as Standard Pascal's ARCTAN)

     LN (NUMBER)

     LOG (NUMBER)

     SQRT (NUMBER)
```

APPENDIX A

DEMONSTRATION PROGRAMS

INTRODUCTION

This appendix presents a graphics program which is fully annotated, both by a narrative explanation and by copious comments in the source text. This program is followed by commentaries on the demonstration programs supplied with Apple Pascal.

A word of caution is in order regarding all of these programs. They are presented so as to give you examples that you can run without any modification, and also study the source text to see how it works. They are not intended to be models of the best possible programming technique; that would be entirely beyond the scope of this manual. They do work, and they do demonstrate ways of doing certain useful things in Apple Pascal. With this caution in mind, use the programs as learning tools. One of the best ways to learn might be to try introducing modifications into one of them.

A FULLY ANNOTATED GRAPHICS PROGRAM

The following demonstration program, PATTERNS, is intended to illustrate some helpful points about Apple Pascal. The program creates pleasant graphics by drawing a triangle on the screen and then repeatedly rotating it by a few degrees and redrawing it. The points of the triangle are always on the edge of an invisible circle of radius 95 (which fills the height of the screen) but apart from that it is a random triangle. The angle by which it is rotated each time it is drawn is also random, though it is always between 3 and 15 degrees.

The color used to draw the triangle is REVERSE, which has intriguing effects when one image is drawn over another and the lines intersect at small angles. Also, as the triangle is repeatedly rotated and redrawn a circular pattern is built up; but eventually the triangle gets rotated back to its original position. When this happens, each new image is exactly superimposed on an old one. Because of the REVERSE color, this erases the old image! When all the old images have been erased, the program clears the screen, generates a new triangle with a different shape, and starts all over.

This repetition continues until the user signals it to halt by pressing any key. The KEYPRESS function, in the APPLESTUFF unit, can be used to find out whether the user has pressed a key. (KEYPRESS is described in Chapter 7.)

The program is given in full, with comments, at the end of this appendix. What follows is a description of how a program like this can be developed. Of course, in real life there are mistakes and false starts. Here, for the sake of learning some principles, we pretend that the development of the program proceeds without a hitch.

This is a fairly complicated program, so we will develop it in sections. First we can write a sketchy outline of the program:

```
BEGIN
  REPEAT
    (*Create a random triangular pattern*);
    THETA:=(*random number from 3 to 15*)
    REPEAT
      (*Rotate the triangle, using the angle THETA*);
      (*Draw the rotated triangle on the screen*)
    UNTIL (*Complete pattern has been erased*)
  UNTIL KEYPRESS
END.
```

To fill in this outline, we begin with a program heading, a USES declaration, some useful constants, two variable declarations, and a skeleton of the inner REPEAT statement:

```
PROGRAM PATTERNS;
USES TURTLEGRAPHICS,APPLESTUFF;

CONST MAXX=28Ø; MAXY=191; (*Maximum X and Y coordinates*)
      RADIUS=95; (*Radius of pattern*)

VAR CYCLES:Ø..2;
    THETA:3..15;

BEGIN
  REPEAT
    (*Create a random triangular pattern*);
    THETA:=(*random number from 3 to 15*);
    CYCLES:=Ø
    REPEAT
      (*Rotate the triangle, using the angle THETA*);
      PENCOLOR(REVERSE);
      (*Draw the rotated triangle on the screen*);
      IF (*the rotated triangle matches the original triangle*)
        THEN CYCLES:=CYCLES+1
    UNTIL CYCLES=2
  UNTIL KEYPRESS
END.
```

The variable CYCLES is a counter for the number of times the triangle has been rotated back to its original position. When CYCLES=1, the circular pattern begins to be erased because each new triangle is drawn in the REVERSE color on top of a previous triangle. When CYCLES=2, the entire pattern has been erased.

We can now begin replacing comments with actual statements. For example, we already have a variable, THETA, which is the number of degrees to rotate the pattern. So it is natural to replace the first comment in the inner REPEAT with a call to a procedure named ROTATE which takes an INTEGER parameter. The value used for the parameter will be the variable THETA. ROTATE will need to be declared; now we have

```
    ...

    PROCEDURE ROTATE(ANGLE:INTEGER);
      (*To be completed...*)

    BEGIN
      REPEAT
        (*Create a random triangular pattern*);
        THETA:=(*random number from 3 to 15*);
        CYCLES:=Ø
        REPEAT
          ROTATE(THETA);
    ...
```

To draw the triangle on the screen, we must first consider how the triangle is represented in memory. We can think of the triangle as three points; how shall we represent a point? A point can be represented by two numbers -- an X and a Y coordinate. Therefore we can define a type POINT, as shown below. Then we can represent the triangle as an array named TRGL, of type POINT. We will also declare a variable C to use as an index for the array TRGL.

```
    ...

    TYPE POINT=RECORD X:Ø..MAXX;
                      Y:Ø..MAXY
               END;

    VAR CYCLES:Ø..2;
        THETA:3..15;
        TRGL:ARRAY[1..3] OF POINT;
        C:1..3;

    ...
```

Assuming that the ROTATE procedure leaves the rotated coordinates in the array TRGL and that it leaves the turtle at the third corner of the triangle, we can use Cartesian graphics to draw the new triangle:

```
        ...

        PENCOLOR(REVERSE);
        FOR C:=1 TO 3 DO MOVETO(TRGL[C].X, TRGL[C].Y);

        ...
```

The remaining comment in the inner REPEAT statement calls for testing whether the rotated triangle matches the original one. To achieve this, assume that when the triangle is first created the coordinates of the third corner are saved in a variable named CORNER. Now we need only test as follows:

```
        ...

        IF TRGL[3]=CORNER THEN CYCLES:=CYCLES+1

        ...
```

At this point, the program is as follows:

```
    PROGRAM PATTERNS;
    USES TURTLEGRAPHICS,APPLESTUFF;

    CONST MAXX=28Ø; MAXY=191; (*Maximum X and Y coordinates*)
          RADIUS=95; (*Radius of pattern*)

    TYPE POINT=RECORD X:Ø..MAXX;
                      Y:Ø..MAXY
               END;

    VAR CYCLES:Ø..2;
        THETA:3..15;
        TRGL:ARRAY[1..3] OF POINT;
        C:1..3;
        CORNER:POINT;

    PROCEDURE ROTATE(ANGLE:INTEGER);
      (*To be completed; must leave new corner coordinates
        in TRGL and leave turtle at third corner.*)

    BEGIN
      REPEAT
        (*Create a random triangular pattern*);
        THETA:=(*random number from 3 to 15*);
        CYCLES:=Ø
        REPEAT
          ROTATE(THETA);
          PENCOLOR(REVERSE);
          FOR C:=1 TO 3 DO MOVETO(TRGL[C].X, TRGL[C].Y);
          IF TRGL[3]=CORNER THEN CYCLES:=CYCLES+1
        UNTIL CYCLES=2
      UNTIL KEYPRESS
    END.
```

The inner REPEAT statement will repeatedly rotate the triangle and draw it, using the REVERSE color, building up a circular pattern on the screen and then erasing it by drawing over it. When the pattern has been erased, the inner REPEAT terminates.

Now we can begin filling in the outer REPEAT. The statements added to the outer REPEAT require another procedure, MAKETRGL, and a function, ARBITRARY.

```
     ...

     FUNCTION ARBITRARY(LOW, HIGH:INTEGER):INTEGER;
       (*To be completed; must return an integer value in the
         range LOW..HIGH.*)

     PROCEDURE MAKETRGL;
       (*To be completed; must leave corner coordinates in TRGL
         and also initialize CORNER with coordinates of third
         corner.*)

     BEGIN
       REPEAT
         MAKETRGL;                  (*Make triangular pattern*)
         THETA:=ARBITRARY(3, 15); (*Choose angle for rotating triangle*)
         CYCLES:=Ø;                 (*Clear the cycle counter*)
         REPEAT
           ROTATE(THETA);
           PENCOLOR(REVERSE);
           FOR C:=1 TO 3 DO MOVETO(TRGL[C].X, TRGL[C].Y);
           IF TRGL[3]=CORNER THEN CYCLES:=CYCLES+1
         UNTIL CYCLES=2
       UNTIL KEYPRESS
     END.
```

The outer REPEAT first calls MAKETRGL. This procedure, still to be defined, chooses three random points on a circle of radius 95 and stores their coordinates in the array TRGL. It also stores the coordinates of the third corner in the variable CORNER.

Next, the function ARBITRARY is used to assign a random value to THETA, the number of degrees to rotate the triangle.

The main program is nearly complete. It remains only to add one new variable named CENTER (of type POINT), and a few initializing statements before the outer REPEAT:

```
     ...

     VAR CYCLES:Ø..2;
         THETA:3..15;
         TRGL:ARRAY[1..3] OF POINT;
         C:1..3;
         CORNER:POINT;
         CENTER:POINT;

     BEGIN
       RANDOMIZE;               (*To get a different sequence each time
                                  program is executed*)
       INITTURTLE;              (*Always do this to use TURTLEGRAPHICS*)
       CENTER.X:=TURTLEX;       (*The turtle is at the center because
       CENTER.Y:=TURTLEY;         INITTURTLE leaves it there.  Save
                                  its coordinates in CENTER.*)
       REPEAT
         ...

         REPEAT
           ...

         UNTIL CYCLES=2
       UNTIL KEYPRESS
     END.
```

The main program is complete, and now we must define the two procedures MAKETRGL and ROTATE and the function ARBITRARY.

The ARBITRARY function is shown in the complete program at the end of this appendix. It is a simplified version of the RAND function given in Chapter 7, in the discussion of the built-in function RANDOM.

RAND handles unacceptable parameters by setting a VAR parameter of type BOOLEAN. ARBITRARY does not need this error-handling capability since it will always be called with constants as parameters. Similarly, RAND has a special provision for the case where the HIGH and LOW parameters are equal; ARBITRARY does not have this provision, and HIGH must be strictly greater than LOW.

In other respects, ARBITRARY is the same as RAND. Incidentally, the complexity of the calculation in both versions is due to the fact that two numbers cannot be added or subtracted if the result would exceed the value MAXINT (32767). The function has to get around this limitation by using the intermediate value MX.

The MAKETRGL procedure must choose three random points on a circle of radius 95, with its center at CENTER. To select three random points, the following method is used:

```
VAR I:1..3;

...

FOR I:=1 TO 3 DO BEGIN
(*Move the turtle to the CENTER point:*)
  MOVETO(CENTER.X, CENTER.Y);

(*Select a random direction to move the turtle away from CENTER,
  and store this angle in an array named DIRECTION; this array will
  need to be declared:*)
  DIRECTION[I]:=ARBITRARY(Ø,359);

(*Turn the turtle in the selected direction:*)
  TURNTO(DIRECTION[I]);

(*Move out to the edge of the circle:*)
  MOVE(RADIUS);

(*Store the turtle's coordinates in the TRGL array:*)
  TRGL[I].X:=TURTLEX;
  TRGL[I].Y:=TURTLEY
END
```

The DIRECTION array will be used by the ROTATE procedure, so it will need to be declared at the beginning of the program -- not within the MAKETRGL procedure.

Since we don't want to draw anything at this point, we set the color to NONE before starting the FOR statement. After three times through the FOR statement, the turtle is at the third corner of the triangle, so we save its position in the CORNER variable for use in the main program. The complete procedure is

```
PROCEDURE MAKETRGL;
  VAR I:1..3;
  BEGIN
    PENCOLOR(NONE);
    FOR I:=1 TO 3 DO BEGIN
      MOVETO(CENTER. , CENTER.Y);
      DIRECTION[I]:=ARBITRARY(Ø, 359);
      TURNTO(DIRECTION[I];
      MOVE(RADIUS);
      TRGL[I].X:=TURTLEX;
      TRGL[I].Y:=TURTLEY
    END;
    CORNER.X:=TURTLEX;
    CORNER.Y:=TURTLEY
  END;
```

The ROTATE procedure works very much like the MAKETRGL procedure, but instead of using random angles it uses the angles found in the DIRECTION array -- after adding ANGLE to each of them and taking the result MOD 36Ø. It stores the resulting points in the TRGL array, but does not change CORNER. The effect is to replace each point in TRGL with a new point created by rotation through ANGLE degrees. The complete ROTATE procedure is

```
PROCEDURE ROTATE(ANGLE:INTEGER);
  VAR I:1..3;
  BEGIN
    PENCOLOR(NONE);
    FOR I:=1 TO 3 DO BEGIN
      MOVETO(CENTER.X, CENTER.Y);
      DIRECTION[I]:=(DIRECTION[I]+ANGLE) MOD 36Ø;
      TURNTO (DIRECTION[I]);
      MOVE(RADIUS);
      TRGL[I].X:=TURTLEX;
      TRGL[I].Y:=TURTLEY
    END
  END;
```

Note that the MOD 36Ø operation is necessary because if the program ran for a long time, the result of DIRECTION[I]+ANGLE could eventually exceed MAXINT and cause a run-time error.

All that remains is to declare the array DIRECTION:

```
DIRECTION:ARRAY[1..3] OF INTEGER;
```

The complete program begins on the following page.

```
PROGRAM PATTERNS;
USES TURTLEGRAPHICS,APPLESTUFF;

(*****************************************************************************)
CONST
(*Maximum X and Y coordinates*)
      MAXX=28Ø; MAXY=191;
(*Radius of pattern*)
      RADIUS=95;

(*****************************************************************************)
TYPE
(*This type stores one set of screen coordinates*)
     POINT=RECORD X:Ø..MAXX;
                  Y:Ø..MAXY
           END;

(*****************************************************************************)
VAR
(*Counter for how many times triangle has been rotated back to its
initial position*)
    CYCLES:Ø..2;
(*Angle for rotating triangle*)
    THETA:3..15;
(*Array to store coordinates of corners of triangle*)
    TRGL:ARRAY[1..3] OF POINT;
(*Index for corners of triangle*)
    C:1..3;
(*Point to store coordinates of one corner of triangle, before any
  rotations*)
    CORNER:POINT;
(*Point to store coordinates of center of screen*)
    CENTER:POINT;
(*Array to store direction angles used to generate triangle*)
    DIRECTION:ARRAY[1..3] OF INTEGER;

(*****************************************************************************)
FUNCTION ARBITRARY (LOW, HIGH:INTEGER):INTEGER;

(*Returns a pseudo-random integer in the range LOW through HIGH.  This
function should only be called with constants as parameters.  HIGH must
be strictly greater than LOW; it must not be equal to LOW.  Also the
difference between HIGH and LOW must not exceed MAXINT.*)

  VAR MX, Z, D: INTEGER;
  BEGIN
    Z:=HIGH-LOW+1;
    MX:=(MAXINT-HIGH+LOW) DIV Z+1;
    MX:=MX*(HIGH-LOW)+(MX-1);
    REPEAT D:=RANDOM UNTIL D <= MX;
    ARBITRARY:=LOW+D MOD Z
  END;
```

```
(*****************************************************************************)
PROCEDURE MAKETRGL;

(*Make a triangle, defined by three randomly chosen points at a distance
RADIUS from the point CENTER.  Choose each point by starting at CENTER,
turning to a random angle, and moving the distance RADIUS.  Store the
angles in DIRECTION, the point coordinates in TRGL, and the third point
(for future reference) in CORNER.  Notice how conveniently this is done
by moving the turtle around with the color NONE.*)

  VAR I:1..3;
  BEGIN
    PENCOLOR(NONE);
    FOR I:=1 TO 3 DO BEGIN
      MOVETO(CENTER.X, CENTER.Y);
      DIRECTION[I]:=ARBITRARY(Ø, 359);
      TURNTO(DIRECTION[I]);
      MOVE(RADIUS);
      TRGL[I].X:=TURTLEX;
      TRGL[I].Y:=TURTLEY
    END;
    CORNER.X:=TURTLEX;
    CORNER.Y:=TURTLEY
  END;

(*****************************************************************************)
PROCEDURE ROTATE(ANGLE:INTEGER);

(*Rotate the triangle defined by point coordinates in TRGL and angles in
DIRECTION, by adding ANGLE to the angles in DIRECTION, taking the
result MOD 36Ø, and using these angles to determine the new corner
coordinates.  Again the turtle is moved around using the color NONE.*)

  VAR I:1..3;
  BEGIN
    PENCOLOR(NONE);
    FOR I:=1 TO 3 DO BEGIN
      MOVETO(CENTER.X, CENTER.Y);
      DIRECTION[I]:=(DIRECTION[I]+ANGLE) MOD 36Ø;
      TURNTO(DIRECTION[I]);
      MOVE(RADIUS);
      TRGL[I].X:=TURTLEX;
      TRGL[I].Y:=TURTLEY
    END
  END;
```

```
(**************************************************************************)
                         (*Main Program*)

BEGIN

(*Do initializations that will not need to be repeated*)

 RANDOMIZE;              (*To get a different sequence each time
                           program is executed*)
 INITTURTLE;             (*Always do this to use TURTLEGRAPHICS*)
 CENTER.X:=TURTLEX;      (*The turtle is at the center because
                           INITTURTLE leaves it there.  Save its
                           coordinates in CENTER.*)
 CENTER.Y:=TURTLEY;

(*The following (outer) REPEAT statement creates a new triangular
pattern each time through.*)

 REPEAT
   MAKETRGL;                 (*Make triangular pattern*)
   THETA:=ARBITRARY(3, 15);  (*Choose angle for rotating triangle*)
   CYCLES:=Ø;                (*Clear the cycle counter*)
```

```
(*The following (inner) REPEAT statement draws the triangle in a new
rotated position each time through.*)

    REPEAT

(*Rotate the triangle.*)

      ROTATE(THETA);

(*Draw the triangle.  This is conveniently done with Cartesian
graphics, since the coordinates are all set up.*)

      PENCOLOR(REVERSE);
      FOR C:=1 TO 3 DO MOVETO(TRGL[C].X, TRGL[C].Y);

(*Now, if the third corner of the triangle matches the CORNER value
saved earlier (by MAKETRGL), then the triangle has been rotated back to
its original position.*)

      IF TRGL[3]=CORNER THEN CYCLES:=CYCLES+1

(*End the repetition if the triangle has returned to its original
position twice.  When this is the case, the pattern has been erased by
being drawn over with the REVERSE color.*)

    UNTIL CYCLES=2

(*End the outer REPEAT statement when a key is pressed.*)

  UNTIL KEYPRESS

END.
```

OTHER DEMONSTRATION PROGRAMS

A set of demonstration programs is supplied with the Pascal System. Although these programs are not fully annotated, they are worth careful study by any student of Pascal. The following are brief descriptions of the programs.

The .TEXT version of each program has been included on diskette APPLE3: so that you can read the program's text into the Editor, to see how the program was written and to try modifications of your own.

DISKETTE FILES NEEDED

The following diskette files allow you to execute the various demonstration programs. The notation xxxxxx stands for the name of a particular demonstraion program.

xxxxxx.CODE	(any diskette, any drive)
SYSTEM.LIBRARY	(boot diskette, boot drive)
SYSTEM.CHARSET	(any diskette, any drive; required if WCHAR or WSTRING used)

One-drive note: Use the Filer to T(ransfer the desired demonstration program's .CODE file to your boot diskette, APPLEØ: or APPLE1:. Then you can X(ecute the program with the boot diskette in the disk drive.

Multi-drive note: You should place your boot diskette, APPLEØ: or APPLE1: , in the boot drive. The demonstration programs are all normally found on diskette APPLE3:. With APPLE3: in any available disk drive, you are ready to X(ecute the demonstration programs.

If you just wish to examine the text version of a demonstration program, there are two ways to proceed:

- For a quick look, put diskette APPLE3: in any available drive, and then use the Filer to T(ransfer the desired program's .TEXT file from APPLE3: to CONSOLE:. To stop the program's listing on the screen, press CTRL-S. Press CTRL-S again to continue.

- To examine the text in more detail, you can E(dit the program's .TEXT file. On one-drive systems, first use the Filer to T(ransfer the program's .TEXT file from APPLE3: to your boot diskette, APPLEØ: or APPLE1:. Then E(dit the file.

If you wish to modify, compile, and execute a new version of a demonstration program, the following diskfiles will be needed:

xxxxxx.TEXT	(any diskette, any drive; required only until read into Editor)
SYSTEM.EDITOR	(any diskette, any drive)
SYSTEM.COMPILER	(any diskette, any drive)
SYSTEM.SYNTAX	(boot diskette, any drive; optional Compiler error messages)
SYSTEM.PASCAL	(boot diskette, boot drive)
SYSTEM.LIBRARY	(boot diskette, boot drive)
SYSTEM.CHARSET	(any diskette, any drive; required if WCHAR or WSTRING used)

One-drive note: Diskette APPLEØ: normally contains all the needed files except the demonstration program's .TEXT file. You should use diskette APPLEØ: as your boot diskette, and T(ransfer the desired demonstration program's .TEXT file to APPLEØ:. Then, with APPLEØ: in the disk drive, you are ready to E(dit and R(un the program.

Two-drive note: Using diskette APPLEØ: as your boot diskette, put APPLEØ: in the boot drive and put APPLE3: in the other drive. You are then ready to E(dit and R(un any program's .TEXT file on APPLE3:.

THE "TREE" PROGRAM

TREE shows the creation of an unbalanced binary tree to sort and retrieve data elements (words, in this case). It lets you specify each new word to be stored in the tree, and then shows you graphically just where the new word was placed in the tree.

When you X(ecute TREE.CODE, you are prompted to

```
ENTER WORD:
```

To quit the program at any time, you can just press the RETURN key in response to this message. To continue, you should type the first word to be sorted (only the first six characters are used). For example, you might type:

```
FLIPPY
```

The program then lists the words entered so far, in alphabetic order.

```
THE WORDS IN ORDER ARE:
FLIPPY
```

No prompting message appears, but you must now press the RETURN key to proceed. When you do, a high-resolution picture is displayed, showing the binary tree as it now exists.

```
     BINARY TREE:

            /
           /
|--------|
| FLIPPY |
|--------|
           \
            \
```

The box represents the binary tree's first "node", or sorting element. The node has two "links" which can point the way to further nodes: the upper link in the display can point to nodes which precede this node alphabetically, while the lower link can point to nodes which follow this node alphabetically.

To continue, press the RETURN key again. Again you are prompted to

```
     ENTER WORD:
```

Suppose you now type

```
     APPLE
```

The program responds

```
     THE WORDS IN ORDER ARE:
     APPLE
     FLIPPY
```

and when you press the RETURN key, another picture of the tree is displayed.

```
     BINARY TREE:
                         /
                        /
             |--------|
            /| APPLE  |
           / |--------|
|--------|              \
| FLIPPY |               \
|--------|
           \
            \
```

This is how the word APPLE is placed in the binary tree. The word APPLE is compared to the word in the first node, FLIPPY. Since APPLE precedes FLIPPY, alphabetically, the search continues by following the first node's upper link. If another node is found at the end of that link, APPLE is compared to the word in that node, and the search continues by following that node's appropriate link. The search continues until, on following an appropriate link, no node is found with which to compare APPLE. At that point on the tree, a new node is created, containing APPLE.

Retrieving the words to list them in alphabetic order is harder to describe, although the algorithm is fairly simple.

1. Starting at the root node, FLIPPY, follow the tree taking only the upper link from each node, until a node is found whose upper link does not connect to a further node. The word in this node is the first word, alphabetically, so print it.

2. Now follow this node's lower link.

   a. If a node is connected to the link, follow the tree taking only the upper link from each node, until a node is found whose upper link does not connect to a further node. Print that node's word as the next one in alphabetic order, and repeat step 2.

   b. If no further node is connected to the link, go back down the tree to the node whose upper link led to this node. Print that node's word as the next one in alphabetic order, and repeat step 2. (If no link or a lower link led to this node, the list is complete.)

Remember, to quit this program just press the RETURN key in response to the message

```
     ENTER WORD:
```

Caution: You must press the RETURN key two times between each word entry (whether or not you wish to see the tree diagrammed). But if you accidentally press RETURN three times, the program is terminated and your list is lost forever.

Program TREE contains examples of the following:

1. Inserting elements into an unbalanced binary tree (INSERTIT)

2. Retrieving elements in order from such a tree (PRINTTREE)

THE "BALANCED" PROGRAM

BALANCED is identical to TREE, except that it stores words by creating a balanced binary tree. It is taken from an example shown on page 215 of the book "Algorithms + Data Structures = Programs", by Nicklaus Wirth (Prentice-Hall, 1976). An AVL-BALANCED BINARY TREE is rearranged after each element insertion to ensure that, of the two branches at any node, one branch is at most one node longer than the other branch. This method of element insertion is slower than for an unbalanced tree, but subsequent retrieval of elements is faster.

Read the description of the TREE demonstration program for details about using this program. New words are added to the BALANCED tree in the same way described for the unbalanced TREE, but the rearrangement of the BALANCED tree following an insertion is more complex. The words are retrieved in alphabetic order identically in the two programs.

THE "CROSSREF" PROGRAM

CROSSREF is an example of a textual cross-reference generator using an unbalanced binary tree to store and sort words. It is taken from an example shown on page 2Ø6 of the book "Algorithms + Data Structures = Programs", by Nicklaus Wirth (Prentice-Hall, 1976).

When you X(ecute CROSSREF.CODE, you are prompted for the name of an

```
     INPUT FILE?
```

Respond by typing the filename of a text file that you wish cross-referenced, on any available diskette. It is not necessary to specify the filename's .TEXT suffix. For example, you might type

```
     APPLEØ:MYSTUFF
```

The program then prompts you to specify a

```
     DESTINATION FILE?
```

for the resulting cross-referenced list. You should respond by typing

```
     CONSOLE:
```

if you want the list to appear on the screen, or

```
     PRINTER:
```

if you want the list to be printed on your printer (which must be connected and turned on).

First, the INPUT text file is displayed on the screen or printed, with each line of text numbered. The words of the text are then stored in alphabetic order in a binary tree, one word to each node. A word is defined as beginning with an alphabetic character and containing all subsequent characters until the next non-alphanumeric character. Finally, the text's words are displayed or printed in alphabetic order, each word followed by the text line numbers where that word appears.

Program CROSSREF contains examples of the following:

1. Set membership (TYPE defines items of the tree structure)

2. Sorting into a binary tree

3. Listing from a binary tree (PRINTTREE, also shows recursion)

For more information about tree-sorting, see the demonstration programs TREE and BALANCED.

THE "SPIRODEMO" PROGRAM

SPIRODEMO demonstrates the basic TURTLEGRAPHICS maneuver: move the pen in a straight line, turn, move again in a straight line, turn again, and so on.

The program lets you specify an ANGLE and a CHANGE, and then draws a pattern on the screen. To make the pattern, SPIRODEMO moves the pen one unit, turns through ANGLE, moves 1+CHANGE, turns ANGLE, moves 1+CHANGE+CHANGE, turns ANGLE, etc.

When you X(ecute SPIRODEMO.CODE, this message appears:

```
     WELCOME TO WHILEPLOT
     ENTER ANGLE Ø TO QUIT.

     ANGLE:
```

If you wish to leave the program at any time, just wait until this prompting message is displayed, and then respond by typing a zero and pressing the RETURN key. If you want to continue, type any positive or negative integer to specify the angle (in degrees) through which you wish the TURTLEGRAPHICS pen to turn between each move. For example, you might respond by typing

```
     89
```

This tells the pen to turn clockwise, slightly less than a right angle between each move. Now you are asked to specify a

```
     CHANGE:
```

Starting with a straight-line pen move of one unit, each subsequent move will increase in length by an amount specified by CHANGE. You must respond by typing a positive integer greater than zero. For example, to make each line one unit longer than the previous line, you would type

```
     1
```

When you press the RETURN key, program SPIRODEMO (alias WHILEPLOT) begins to draw its design on the screen, using the parameters that you specified.

On completion of the design, the program continues to display the design until you press any key on the Apple's keyboard. Just press the Apple's spacebar, and the original prompt message will replace the design on the screen. You are then ready to specify a new CHANGE and DISTANCE for another design (or specify an ANGLE of zero to quit the program).

Caution: This program dies if the first character of an ANGLE or CHANGE response is not a plus sign, a minus sign, or a numeric digit.

Program SPIRODEMO contains examples of the following:

1. Using the TURTLEGRAPHICS unit, including the KEYPRESS function

2. Reading the keyboard buffer without echoing on the screen

THE "HILBERT" PROGRAM

HILBERT shows an historically famous example of recursion, using a space-filling design to create an attractive display on the screen.

You can determine the density of the space-filling design by specifying an integer ORDER from 1 through 7.

When you X(ecute HILBERT.CODE, this message appears:

```
ENTER ORDER Ø TO QUIT.

ORDER:
```

If you wish to quit the program at any time, wait until this message appears, and then type a zero. If you wish to continue, you must type an integer from 1 through 7. An ORDER of 1 fills the space most "loosely", taking barely one repetition of the design to fill the screen. Each higher order fills the screen more and more densely, by repeating the basic design on a smaller and smaller scale. Order 7 fills the screen to solid white, and takes quite a long time doing it. There is no way to stop a display while it is being created, except to press the RESET key. To get the idea, respond by typing

```
4
```

On completion of the design, the program continues to display the design until you press any key on the Apple's keyboard. Just press the Apple's spacebar, and the original prompt message will replace the design on the screen. You are then ready to specify a new ORDER for another design (or specify an ORDER of zero to quit the program).

Caution: This program is terminated if the ORDER response is not a numeric digit from 1 through 7.

THE "GRAFDEMO" PROGRAM

GRAFDEMO is a collection of interesting graphical displays generated by a number of very useful procedures.

The program runs without any interaction; just watch the pretty pictures and then study GRAFDEMO.TEXT to see examples of how these things can be done using TURTLEGRAPHICS. You may even find it handy to use some of GRAFDEMO's procedures directly, in your own programs.

When you X(ecute GRAFDEMO.CODE, this unusual message appears:

```
PRESS ANY KEY TO QUIT.
PLEASE WAIT WHILE CREATING BUTTERFLY
```

Just wait; soon you will see butterflies and many other graphical marvels. Pressing any key on the Apple keyboard will terminate this program on completion of whichever display is currently being created.

Program GRAFDEMO contains examples of the following:

1. Using TURTLEGRAPHICS to draw frames, crosshatching, etc.

2. Creation of an array (BUTTER) for use by procedure DRAWBLOCK

3. Handling of a procedure that is too long, by breaking it into smaller parts (BUTTER) and calling those parts from another procedure (INITBUTTER)

THE "GRAFCHARS" PROGRAM

GRAFCHARS shows the characters found in the file SYSTEM.CHARSET, and their use from TURTLEGRAPHICS. The program runs without interaction.

When you X(ecute GRAFCHARS.CODE, this message appears:

```
PRESS RETURN FOR MORE...
```

From here on, each time you press the Apple's RETURN key another display is placed on the screen. The first display shows all the characters available in SYSTEM.CHARSET . When you have examined any display to your satisfaction, just press the RETURN key again to go on to the next display.

Program GRAFCHARS contains examples of the following:

1. All the upper-case, lower-case, and special characters available through TURTLEGRAPHICS

2. Use of TURTLEGRAPHICS' WCHAR and WSTRING functions

3. How to put a border around a string (BOXSTRING)

4. Use of CHARMODE to keep the characters' boundaries from interfering with the background

THE "DISKIO" PROGRAM

DISKIO shows a sample use of random-access disk files, with terminal-independent output.

Note: This program is NOT a real application, and it is definitely NOT a data-base manager. Its only purpose is to demonstrate some of the principles that would be involved in writing a real file-handling program.

When you X(ecute DISKIO.CODE, you are asked to specify a

```
     FILE NAME:
```

You should type a valid disk-file identifier. For example, you might respond by typing

```
     APPLEØ:MYFILE.TEXT
```

The program looks on the specified diskette (or the default diskette) for a file with the specified filename. If an existing file by that name is found, it is opened and the main program command prompt line is displayed. If no file by that name is found, the program asks if it should

```
     START A NEW FILE?
```

If you type N for No, you will again be asked to type a FILE NAME. There is no exit from the program at this point except by successfully opening a file or by pressing the RESET key. If you type Y for Yes, the program asks

```
     RESERVE HOW MANY RECORDS?
```

Respond by typing an integer that specifies the number of records your new file will initially contain. For example, if you type

```
     6
```

your new file will start out containing seven records, numbered Ø through 6.

Now the program's main command prompt line appears on the screen:

```
     V(IEW  C(HANGE  N(EXT  F(ILE  Q(UIT
```

Typing a V for V(iew causes this message to appear:

```
     VIEW WHICH RECORD?
```

You should respond by typing a number from zero through the maximum record number in your file. For instance, typing

```
     5
```

lets you view the contents of record number 5.

If you then wish to view the contents of the next record, type N for N(ext. In this way, you can look at as many records as you wish.

Typing a C for C(hange causes this message to appear:

```
     CHANGE WHICH RECORD?
```

Again, you should respond by typing a number from zero through the maximum record number in your file. For instance, typing

```
     5
```

lets you change the contents of record number 5. To change an entry, just start typing. To leave an entry as it is, and go on to the next entry, just press the RETURN key.

If you then wish to change the contents of the next record, type N for N(ext. In this way, you can change as many records as you wish.

If the N(ext command takes you beyond the last record specified for your file, the program will attempt to extend the file by appending additional records. This is possible if

1. there is room for the record in the current last block of the file, or

2. the next contiguous block on the diskette is available for use by this file.

If it is not possible to extend your file, a message appears to inform you of the problem. You can then type Q to Q(uit this program, enter the Filer, and move files on the diskette until your file has a few free blocks immediately following it. (Use the Filer's E(xtended List command to see the locations of free blocks.) Then you are ready to X(ecute DISKIO again, and extend your file with additional records.

Typing F for F(ile, in response to the main command prompt line, lets you start a new file or reopen another old file. As at the beginning, you are asked for a

```
     FILE NAME:
```

Again, there is no exit from this part of the program except to give a successful filename or to press the RESET key.

Program DISKIO contains examples of the following:

1. Terminal-independent output, by reading the file SYSTEM.MISCINFO and using the terminal setup parameters found there (GETCRTINFO)
2. Bullet-proof character input (GETCHAR)
3. Bullet-proof string input, with defaults
4. Use of random-access disk files and system procedure SEEK
5. How to extend a diskette file in place.

APPENDIX B
TABLES

TABLE 1: EXECUTION ERRORS

Ø	System error	FATAL
1	Invalid index, value out of range (XINVNDX)	
2	No segment, bad code file (XNOPROC)	
3	Procedure not present at exit time (XNOEXIT)	
4	Stack overflow (XSTKOVR)	
5	Integer overflow (XINTOVR)	
6	Divide by zero (XDIVZER)	
7	Invalid memory reference <bus timed out> (XBADMEM)	
8	User break (XUBREAK)	
9	System I/O error (XSYIOER)	FATAL
1Ø	User I/O error (XUIOERR)	
11	Unimplemented instruction (XNOTIMP)	
12	Floating point math error (XFPIERR)	
13	String too long (XS2LONG)	
14	Halt, Breakpoint (without debugger in core) (XHLTBPT)	
15	Bad Block	

All FATAL errors require that the system be rebooted. In some cases the system will reboot automatically, and in other cases you will have to reboot it. All other errors cause the system to re-initialize itself.

TABLE 2: I/O ERRORS (IORESULT VALUES)

Ø No error

1 Diskette has bad Block: parity error (CRC). (Not used on the Apple.)

2 Bad device (volume) Number

3 Bad Mode: illegal operation. (For example, an attempt to read from PRINTER:.)

4 Undefined hardware error. (Not used on the Apple.)

5 Lost device: device is no longer on-line, after successfully starting an operation using that device.

6 Lost file: file is no longer in the diskette directory, after successfully startng an operation using that file.

7 Bad title: illegal file name. (For example, filename is more than 15 characters long.)

8 No room: insufficient space on the specified diskette. (Files must be stored in contiguous diskette blocks.)

9 No device: the specified volume is not on line

1Ø No file: The specified file is not in the directory of the specified volume.

11 Duplicate file: attempt to rewrite a file when a file of that name already exists.

12 Not closed: attempt to open an open file.

13 Not open, attempt to access a closed file.

14 Bad format, error in reading real or integer. (For example, your program expects an integer input but you typed a letter.)

15 Ring buffer overflow: characters are arriving at the Apple faster than the input buffer can accept them.

16 Write-protect error: the specified diskette is write-protected.

64 Device error: failed to complete a read or write correctly (bad address or data field on diskette).

See Chapter 3 for description of the built-in function IORESULT.

TABLE 3: RESERVED WORDS

These are words that have fixed meanings in Pascal. You can never use them as identifiers without causing a compiler error. The next two tables list some more words you should not use as identifiers.

STANDARD PASCAL RESERVED WORDS

AND
ARRAY
BEGIN
CASE
CONST
DIV
DO
DOWNTO
ELSE
END
FILE
FOR
FORWARD
FUNCTION
GOTO
IF
IN
LABEL

MOD
NIL
NOT
OF
OR
PACKED
PROCEDURE
PROGRAM
RECORD
REPEAT
SET
THEN
TO
TYPE
UNTIL
VAR
WHILE
WITH

ADDITIONAL APPLE PASCAL RESERVED WORDS

EXTERNAL
IMPLEMENTATION
INTERFACE
SEGMENT
UNIT
USES

TABLE 4: PREDEFINED IDENTIFIERS

These are the identifiers of the built-in procedures and functions and the predefined types and variables of Apple Pascal. The list does not include those identifiers that are declared or defined in the special UNITs supplied for the Apple (see next table). If you declare or define one of these identifiers in your program, no error will result but you will lose the capability of the corresponding built-in or predefined entity.

With each identifier, a code is shown in {brackets} to indicate what kind of object the identifier represents. The codes are

{p} PROCEDURE	{i} INTEGER FUNCTION
{b} BOOLEAN FUNCTION	{r} REAL FUNCTION
{t} TYPE	{c} CHAR FUNCTION
{k} CONSTANT	{f} FILE
{s} STRING FUNCTION	{-} OTHER

ABS {r}
BLOCKREAD {i}
BLOCKWRITE {i}
BOOLEAN {t}
CHAR {t}
CHR {c}
CLOSE {p}
CONCAT {s}
COPY {s}
DELETE {p}
EOF {b}
EOLN {b}
EXIT {p}
FALSE {k}
FILLCHAR {p}
GET {p}
GOTOXY {p}
HALT {p}
INPUT {f}
INSERT {p}
INTEGER {t}
INTERACTIVE {t}

IORESULT {i}
KEYBOARD {f}
LENGTH {i}
MARK {p}
MAXINT {k}
MEMAVAIL {i}
MOVELEFT {p}
MOVERIGHT {p}
NEW {p}
ODD {b}
ORD {i}
OUTPUT {f}
PAGE {p}
POS {i}
PRED {-}
PUT {p}
PWROFTEN {r}
READ {p}
READLN {p}
REAL {t}
RELEASE {p}
RESET {p}

REWRITE {p}
ROUND {i}
SCAN {i}
SEEK {p}
SIZEOF {i}
SQR {r}
STR {s}
STRING {t}
SUCC {-}
TEXT {t}
TREESEARCH {i}
TRUE {k}
TRUNC {i}
UNITBUSY {b}
UNITCLEAR {p}
UNITREAD {p}
UNITWAIT {p}
UNITWRITE {p}
WRITE {p}
WRITELN {p}

TABLE 5: IDENTIFIERS DECLARED IN SUPPLIED UNITS

These identifiers are effectively declared or defined only if your program USES their respective UNITs. If your program USES a UNIT and you attempt to declare or define one of the identifiers belonging to that UNIT, you will get a compiler error message 1Ø1: "Identifier declared twice." However if your program doesn't USE a particular UNIT you can make free use of the identifiers of that UNIT.

With each identifier, a code is shown in {brackets} to indicate what kind of object the identifier represents. The codes are

{p} PROCEDURE
{b} BOOLEAN FUNCTION
{t} TYPE
{i} INTEGER FUNCTION
{r} REAL FUNCTION

TURTLEGRAPHICS UNIT

CHARTYPE {p}
DRAWBLOCK {p}
FILLSCREEN {p}
GRAFMODE {p}
INITTURTLE {p}
MOVE {p}
MOVETO {p}
PENCOLOR {p}
SCREENBIT {b}
SCREENCOLOR {t}
TEXTMODE {p}
TURN {p}
TURNTO {p}
TURTLEANG {i}
TURTLEX {i}
TURTLEY {i}
VIEWPORT {p}
WCHAR {p}
WSTRING {p}

APPLESTUFF UNIT

BUTTON {i}
KEYPRESS {b}
NOTE {p}
PADDLE {i}
RANDOM {i}
RANDOMIZE {p}
TTLOUT {p}

TRANSCEND UNIT

ATAN {r}
COS {r}
EXP {r}
LN {r}
LOG {r}
SIN {r}
SQRT {r}

TABLE 6: COMPILER ERROR MESSAGES

When the Pascal Compiler discovers an error in your program, it reports that error immediately, by error number. If you then enter the Editor to fix that error, a more complete error message is given, taken from the boot diskette file SYSTEM.SYNTAX . If you remove the file SYSTEM.SYNTAX from the boot diskette, errors will be reported by number, only.

The Pascal Compiler error message corresponding to each error number is given in the table below. Some people will prefer to gain some additional space on their boot diskette, by removing SYSTEM.SYNTAX and using this table instead. You can also print your own copy of this table by T(ransferring the file SYSTEM.SYNTAX to a printer.

1: Error in simple type
2: Identifier expected
3: 'PROGRAM' expected
4: ')' expected
5: ':' expected
6: Illegal symbol (possibly missing ';' on line above)
7: Error in parameter list
8: 'OF' expected
9: '(' expected
1Ø: Error in type
11: '[' expected
12: ']' expected
13: 'END' expected
14: ';' expected (possibly on line above)
15: Integer expected
16: '=' expected
17: 'BEGIN' expected
18: Error in declaration part
19: Error in <field-list>
2Ø: '.' expected
21: '*' expected
22: 'Interface' expected
23: 'Implementation' expected
24: 'Unit' expected

5Ø: Error in constant
51: ':=' expected
52: 'THEN' expected
53: 'UNTIL' expected
54: 'DO' expected
55: 'TO' or 'DOWNTO' expected in for statement
56: 'IF' expected
57: 'FILE' expected
58: Error in <factor> (bad expression)
59: Error in variable

1Ø1: Identifier declared twice

```
1Ø2: Low bound exceeds high bound
1Ø3: Identifier is not of the appropriate class
1Ø4: Undeclared identifier
1Ø5: Sign not allowed
1Ø6: Number expected
1Ø7: Incompatible subrange types
1Ø8: File not allowed here
1Ø9: Type must not be real
11Ø: <tagfield> type must be scalar or subrange
111: Incompatible with <tagfield> part
112: Index type must not be real
113: Index type must be a scalar or a subrange
114: Base type must not be real
115: Base type must be a scalar or a subrange
116: Error in type of standard procedure parameter
117: Unsatisfied forward reference
118: Forward reference type identifier in variable declaration
119: Re-specified parameters not OK for a forward declared procedure
12Ø: Function result type must be scalar, subrange or pointer
121: File value parameter not allowed
122: A forward declared function's result type can't be re-specified
123: Missing result type in function declaration
124: F-format for reals only
125: Error in type of standard procedure parameter
126: Number of parameters does not agree with declaration
127: Illegal parameter substitution
128: Result type does not agree with declaration
129: Type conflict of operands
13Ø: Expression is not of set type
131: Tests on equality allowed only
132: Strict inclusion not allowed
133: File comparison not allowed
134: Illegal type of operand(s)
135: Type of operand must be boolean
136: Set element type must be scalar or subrange
137: Set element types must be compatible
138: Type of variable is not array
139: Index type is not compatible with the declaration
14Ø: Type of variable is not record
141: Type of variable must be file or pointer
142: Illegal parameter solution
143: Illegal type of loop control variable
144: Illegal type of expression
145: Type conflict
146: Assignment of files not allowed
147: Label type incompatible with selecting expression
148: Subrange bounds must be scalar
149: Index type must be integer
15Ø: Assignment to standard function is not allowed
151: Assignment to formal function is not allowed
152: No such field in this record
153: Type error in read
154: Actual parameter must be a variable
155: Control variable cannot be formal or non-local
```

```
156: Multidefined case label
157: Too many cases in case statement
158: No such variant in this record
159: Real or string tagfields not allowed
16Ø: Previous declaration was not forward
161: Again forward declared
162: Parameter size must be constant
163: Missing variant in declaration
164: Substitution of standard proc/func not allowed
165: Multidefined label
166: Multideclared label
167: Undeclared label
168: Undefined label
169: Error in base set
17Ø: Value parameter expected
171: Standard file was re-declared
172: Undeclared external file
174: Pascal function or procedure expected

182: Nested units not allowed
183: External declaration not allowed at this nesting level
184: External declaration not allowed in interface section
185: Segment declaration not allowed in unit
186: Labels not allowed in interface section
187: Attempt to open library unsuccessful
188: Unit not declared in previous 'Uses' declaration
189: 'Uses' not allowed at this nesting level
19Ø: Unit not in library
191: No private files
192: 'Uses' must be in interface section
193: Not enough room for this operation
194: Comment must appear at top of program
195: Unit not importable

2Ø1: Error in real number - digit expected
2Ø2: String constant must not exceed source line
2Ø3: Integer constant exceeds range
2Ø4: 8 or 9 in octal number

25Ø: Too many scopes of nested identifiers
251: Too many nested procedures or functions
252: Too many forward references of procedure entries
253: Procedure too long
254: Too many long constants in this procedure
256: Too many external references
257: Too many externals
258: Too many local files
259: Expression too complicated

3ØØ: Division by zero
3Ø1: No case provided for this value
3Ø2: Index expression out of bounds
3Ø3: Value to be assigned is out of bounds
3Ø4: Element expression out of range
```

35Ø: No data segment allocated
351: Segment used twice
352: No code segment allocated
353: Non-intrinsic unit called from intrinsic unit
354: Too many segments for the segment dictionary

398: Implementation restriction
399: Implementation restriction
4ØØ: Illegal character in text
4Ø1: Unexpected end of input
4Ø2: Error in writing code file, not enough room
4Ø3: Error in reading include file
4Ø4: Error in writing list file, not enough room
4Ø5: Call not allowed in separate procedure
4Ø6: Include file not legal
4Ø7: Too many libraries

TABLE 7: ASCII CHARACTER CODES

Code		Char	Code		Char	Code		Char	Code		Char
Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
Ø	ØØ	NUL	32	2Ø	SP	64	4Ø	@	96	6Ø	`
1	Ø1	SOH	33	21	!	65	41	A	97	61	a
2	Ø2	STX	34	22	"	66	42	B	98	62	b
3	Ø3	ETX	35	23	#	67	43	C	99	63	c
4	Ø4	EOT	36	24	$	68	44	D	1ØØ	64	d
5	Ø5	ENQ	37	25	%	69	45	E	1Ø1	65	e
6	Ø6	ACK	38	26	&	7Ø	46	F	1Ø2	66	f
7	Ø7	BEL	39	27	'	71	47	G	1Ø3	67	g
8	Ø8	BS	4Ø	28	(	72	48	H	1Ø4	68	h
9	Ø9	HT	41	29	)	73	49	I	1Ø5	69	i
1Ø	ØA	LF	42	2A	*	74	4A	J	1Ø6	6A	j
11	ØB	VT	43	2B	+	75	4B	K	1Ø7	6B	k
12	ØC	FF	44	2C	,	76	4C	L	1Ø8	6C	l
13	ØD	CR	45	2D	-	77	4D	M	1Ø9	6D	m
14	ØE	SO	46	2E	.	78	4E	N	11Ø	6E	n
15	ØF	SI	47	2F	/	89	4F	O	111	6F	o
16	1Ø	DLE	48	3Ø	Ø	8Ø	5Ø	P	112	7Ø	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	5Ø	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
2Ø	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	12Ø	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	9Ø	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	6Ø	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
3Ø	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

APPENDIX C

ADDITIONAL DETAILS OF TEXT I/O

Here are some facts about READ and READLN that you need to know if you do not follow the suggestions in the "Introduction to Text I/O" section of Chapter 3. In particular, these facts are important if you mix reading and writing operations on the same diskette textfile. You may also need to know exactly when EOLN and EOF become true with READLN and with numeric variables.

Note that for mixed reading and writing, the rules given below are more straightforward for INTERACTIVE file than for TEXT files.

After READ with a CHAR variable and an INTERACTIVE file:

- The file buffer variable contains the character that was READ, unless EOLN or EOF is true.

- If the next I/O operation is a PUT, WRITE, or WRITELN, it affects the character after the one that was READ.

- EOF is true if the character READ was the end-of-file character. In this case the value of the file buffer variable is undefined.

- EOLN is true if the character READ was the end-of-line character. In this case the file buffer variable contains a space.

- EOLN is also true if EOF is true.

After READ with a CHAR variable and a TEXT file:

- The file buffer variable contains the character after the character that was READ, unless EOLN or EOF is true.

- If the next I/O operation is a PUT, WRITE, or WRITELN, it affects the second character after the one that was READ.

- EOF is true if the character READ was the last character in the file (not counting the end-of-file character). In this case the value of the file buffer variable is undefined.

- EOLN is true if the character READ was the last character on the line (not counting the end-of-line character). In this case the file buffer variable contains a space.

- EOLN is also true if EOF is true.

After READ with a numeric variable and a TEXT or INTERACTIVE file:

- The file buffer variable contains the character after the last character of the numeric string that was READ, unless EOLN or EOF is true.

- If the next I/O operation is a PUT, WRITE, or WRITELN, it affects the second character after the last character of the numeric string.

- EOF is true if the last character of the numeric string was the last character in the file (not counting the end-of-file character). In this case the value of the file buffer variable is undefined.

- EOLN is true if the last character of the numeric string was the last character on the line (not counting the end-of-line character). In this case the file buffer variable contains a space.

- EOLN is also true if EOF is true.

After READ with a STRING variable and a TEXT or INTERACTIVE file:

- The file buffer variable contains a space which represents the end-of-line character at the end of the line, unless EOF is true.

- If the next I/O operation is a PUT, WRITE, or WRITELN, it affects the first character on the next line.

- EOF is true if the line READ was the last line in the file. In this case the value of the file buffer variable is undefined.

- EOLN is always true.

After READLN with any variable and an INTERACTIVE file

- The file buffer variable contains a space which represents the end-of-line character at the end of the line, unless EOF is true.

- If the next I/O operation is a PUT, WRITE, or WRITELN, it affects the first character on the next line.

- EOF is true if the line READ was the last line in the file. In this case the value of the file buffer variable is undefined.

- EOLN is never true.

After READLN with any variable and a TEXT file

- The file buffer variable contains the first character on the next line, unless EOLN or EOF is true.

- If the next I/O operation is a PUT, WRITE, or WRITELN, it affects the second character on the next line.

- EOF is true if the line READ was the last line in the file. In this case the value of the file buffer variable is undefined.

- EOLN is true only when EOF is true.

APPENDIX D

ONE-DRIVE STARTUP

This appendix is a tutorial session to get you started using the Language System with Pascal, on an Apple II with one diskette drive. If your system has two or more diskette drives, please skip this appendix and read Appendix E instead.

EQUIPMENT YOU WILL NEED

You should have the following:

1. Your 48K Apple computer, with a Language Card installed, and one disk drive attached to the connector marked "DRIVE 1" on the disk controller card. The disk controller card must have the new PROMs, P5A and P6A (which came with the Language System), and must be installed in the Apple's peripheral device slot 6.

2. A TV set or video monitor properly connected to your Apple.

3. The following Language System diskettes:

   a. APPLEØ:
   b. APPLE1:
   c. APPLE2:
   d. APPLE3:
   e. A blank diskette
   f. Another blank diskette

The diskettes marked "APPLE1:" and "APPLEØ:" are needed to start the system. The diskette marked "APPLE2:" adds some extra features to the system (the Assembler and the Linker). You will not need the diskette marked "APPLE2:" until later (many users of single-drive systems will never need it). The diskette marked "APPLE3:" contains a number of useful utility programs, and some interesting demonstrations; Appendix A of this manual explains these demonstrations.

Your Apple and its TV or monitor should be plugged in. Turn on the TV now, so that it can warm up; but leave the Apple turned off.

THE TWO-STEP STARTUP

There are two steps to starting Apple Pascal running on your system.

STEP ONE OF STARTUP

First insert the diskette marked APPLE1: in the disk drive. If you are not familiar with handling diskettes, see the manuals that came with your disk drives. Diskettes must be treated correctly if they are to last.

Close the door to the disk drive, and turn on the Apple. The rest of Step One is automatic. First, the message

```
                    APPLE II
```

appears at the top of your TV or monitor screen, and the disk drive's "IN USE" light comes on. The disk drive emits a whirring, zickking sound that is as pleasant as a cat's purring, since it lets you know that everything is working. The screen lights up for an instant with a display of black at-signs (@) on a white background, then goes black again. Next, the disk drive stops entirely for a moment; then it whirrs some more. Finally, the message

```
          WELCOME  APPLE1,  TO
          U.C.S.D.  PASCAL SYSTEM II.1
          CURRENT DATE IS 26-JUL-79
```

appears (the date will be different), followed in a second or so by a line at the top of the screen:

```
     COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(IN
```

This line at the top of the screen is called a "prompt line". When you see this prompt line, you know that your Apple computer is running the Apple Pascal system.

If you just wish to edit text and programs, or if you wish to run previously compiled programs, you may stop now. At this point, your system can do most of the things you will normally want to do in Apple Pascal, except for compiling new programs that you write.

However, if you also wish to compile programs that you write, in order to run them, you should proceed to Step Two of the startup procedure.

STEP TWO OF STARTUP

Remove the diskette marked APPLE1: from the disk drive, and insert the one marked APPLEØ: . Close the door to the drive and press the key marked RESET , in the upper right corner of the Apple's keyboard.

The at-signs come back for an instant, and the disk drive whirrs and completely stops for a second, then whirrs some more. The whole process takes about 16 seconds. Finally, the message

```
          WELCOME  APPLEØ,  TO
          U.C.S.D.  PASCAL SYSTEM II.1
          CURRENT DATE IS 26-JUL-79
```

appears (the date will be different), followed in a second or so by the prompt line at the top of the screen

```
          COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(IN
```

Again, this prompt line lets you know that your Apple computer is running the Apple Pascal System.

After completing Step Two of the startup procedure, your system can do all the things you will normally want to do in Apple Pascal: filing, editing, running....and compiling. However, diskette APPLEØ: is missing one file that is needed for the initial startup when you first turn the Apple's power on. That is why you must go through Step One of the startup procedure before going on to Step Two.

CHANGING THE DATE

The date that comes on the diskette will not be correct. It is a good habit to reset the date the first time you use the Apple Pascal System on any given day. It only takes a few seconds. Press F on the keyboard (without pressing the RETURN key or any other keys). The screen goes blank, and then this line appears at the top:

```
FILER: G, S, N, L, R, C, T, D, Q
```

This is a new prompt line. Prompt lines are named after their first word. The prompt line you first saw was the "COMMAND" prompt line. This one is the "FILER" prompt line. Sometimes we say that you are "in the Filer" when this line is at the top of the screen. Each of the letters on the prompt line represents a task that you can ask the system to do. For example, to change the date, press D (again, just type the single key, without pressing RETURN or any other key).

When you do, another message is put on the screen. It says:

```
DATE SET: <1..31>-<JAN..DEC>-<ØØ..99>
TODAY IS 26-JUL-79
NEW DATE ?
```

It doesn't really mean that today is 26-JUL-79 (or whatever date your screen shows), but that the Apple THINKS that is today's date. Since it isn't, you can change the date to be correct. The correct form for typing the date is shown on the second line of the message: one or two digits giving the day of the month, followed by a minus sign, followed by the first three letters of the name of the month, followed by another minus sign, followed by the last two digits of the current year. Then press the key marked RETURN .

If the month and year are correct (as they will often be, when you change the date) all you have to do is type the correct day of the month, and press the RETURN key. The system will assume that you mean to keep the same month and year displayed by the message. If you type a day and a month, the system will assume you mean to keep only the year the same.

Go ahead and make the date correct. This is your first interaction with the system, and is typical of how the system is used. In general, at the top of the screen there will usually be a prompt line which represents several choices of action. When you type the first letter of one of the choices, either you will be shown a new prompt line giving a further list of choices, or else the system will carry out the desired action directly. If you type a letter that does not correspond to one of the choices, the prompt line blinks but otherwise nothing happens. Remember to type only a single letter to indicate your choice; it is not necessary to press the RETURN key afterward.

Sometimes, as when setting the date, you are asked to type a response of several characters. You tell the system that your response is complete by pressing the RETURN key. If you make a typing error before pressing the RETURN key, you can back up and correct the error by pressing the left-arrow key. You should experiment by making deliberate errors in entering a date, and then erasing the errors with the left-arrow key.

One further note. Normally, your new date is saved on the diskette, so the system "remembers" this date the next time you turn the Apple on. However, since you are using the write-protected diskettes that came with your Language System, your new date was not permanently saved. The next time you turn the Apple off, the new date will be "forgotten". By the end of this session, you will have made backup copies of the Language System diskettes. From then on, you will use these copies, which are not write-protected, and your date changes will be saved correctly.

MAKING BACKUP DISKETTE COPIES

WHY WE MAKE BACKUPS

Ask yourself this question: What would happen to your system if you were to lose or damage one of the system diskettes (APPLEØ:, APPLE1:, APPLE2:, or APPLE3:)? It would be as bad as losing your Apple, as far as your being able to use Pascal.

These diskettes are quite precious. The first thing you should do, therefore, is to make backup copies of them. Afterward, you should never use the originals, but put them someplace where the temperature is moderate, where there is no danger of them getting wet, and where such diskette destroyers as dogs, dirt, children, and magnetic fields cannot get at them.

A truly cautious person will keep on hand two backup copies of each original. That way, you will need to use an original only in the very rare case when both of its backup copies are lost (when one copy is lost or damaged, another backup copy is made from the surviving backup copy). If your backups were damaged or erased while in use, find out why they were destroyed before inserting your only surviving copy. Using diskettes for which you have backups, repeat the procedure that destroyed the first diskettes; if you can't figure out what the problem is, take your system to the dealer to make sure it is working correctly.

HOW WE MAKE BACKUPS

The Apple Pascal system can copy all the information from one diskette (or any portion of the information) onto another diskette. But the system cannot store information on a new diskette, just as that diskette comes from the computer store. Therefore, the system is supplied with a program that allows you to take any 5-inch floppy diskette and "format" it so that it will work with the Apple Pascal system.

Incidentally, this is one of the nice little things about the Apple system: ANY high-quality 5-inch floppy diskette (Apple recommends diskettes made by Dysan Corporation) will work on it. Some systems require you to have "1Ø sector" or "15 sector"or "soft sectored" diskettes. The Apple doesn't care, it takes any of these kinds of diskettes, and (through the FORMATTER program) makes them into the kind of diskette it needs.

If you have been following this discussion by carrying out the instructions on your Apple, the FILER prompt line should be showing at the top of the screen:

```
FILER: G, S, N, L, R, C, T, D, Q
```

Type Q on the keyboard to Quit the Filer.

GETTING THE BIG PICTURE

When you Quit the Filer, the disk whirrs, and you see the COMMAND prompt line again:

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(IN
```

There is actually more of this prompt line, off to the right of your TV or monitor. To see the rest of the screen, hold down the key marked CTRL and, while holding it down, press the A key right alongside it. (Or, to be brief, we say: "press CTRL-A".)

You now see

```
K, X(ECUTE, A(SSEM, D(EBUG,?
```

This is simply the rest of the line that began "COMMAND:". All together, the full prompt line would look like this:

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(INK, X(ECUTE, A(SSEM, D(EBUG,?
```

The Apple Pascal system displays information on a "screen" that is 8Ø characters wide, but your TV or monitor shows only the leftmost 4Ø characters or the rightmost 4Ø characters at any one time. You use the CTRL-A trick whenever you wish to see if there is more stuff on the other "half" of the screen. Repeated pressing of CTRL-A flips back and forth between the left half of the screen and the right half. Also, sometimes the TV display will seem to be blank. This might mean that you are just staring at the empty right half of the screen. Before you come to the conclusion that something is wrong, always try CTRL-A. You get back to the left side of the screen by typing CTRL-A again, and you might find that everything is OK after all.

Summary of this digression: The screen is really twice as wide as it looks. To flip from the left side to the right side or back again, you type CTRL-A.

FORMATTING NEW DISKETTES

When the COMMAND prompt line is showing at the top of the screen, remove your system diskette (APPLE1: or APPLEØ:) from the disk drive and place the diskette APPLE3: in the drive. This has to be done because the FORMATTER program is on APPLE3: . Now, type

```
X
```

and the screen responds:

```
EXECUTE WHAT FILE?
```

You type

```
APPLE3: FORMATTER
```

and press the RETURN key. The disk whirrs a bit and the screen says:

```
APPLE DISK FORMATTER PROGRAM
FORMAT WHICH DISK (4, 5, 9..12) ?
```

Now comes a grand session. Take all the new, blank diskettes that you are going to use with the Apple Pascal System (but not, of course, any diskettes that have precious information on them, such as the diskettes that came with the Apple Pascal System) and place them in a pile. Their labels should be blank. Make sure that you don't have any diskettes with data in a non-Pascal format, such as BASIC diskettes: the Apple Pascal system will be unable to read them, and will regard them as blank, erasing any old information in the formatting process.

Remove the diskette APPLE3: from the disk drive, and place one of the blank diskettes into the drive. Type

```
          4
```

and press the RETURN key.

If the diskette in the drive has already been formatted, you will receive a warning. For example, if you have left APPLE3: in the drive you will be warned with the message

```
          DESTROY DIRECTORY OF APPLE3 ?
```

At this point you can type

```
          N
```

(which stands for "No") without pressing the RETURN key, and your diskette will not be destroyed.

Let's assume that you have placed a new, unformatted diskette in the disk drive. Then you will not get any warning, but the Apple will place this message on the screen:

```
          NOW FORMATTING DISKETTE IN DRIVE 4
```

The drive will make some clickings and buzzings and begin to whirr and zick. The process takes about 32 seconds. When formatting is complete, the screen again shows the message

```
          FORMAT WHICH DISK (4, 5, 9..12) ?
```

Now you have a formatted diskette. We suggest that you write the word "Pascal" in small letters at the top of the diskette's label, using a marking pen. Do not use a pencil or ballpoint pen, as the pressure may damage the diskette. The label will let you know that the diskette is formatted for use with the Apple Pascal system, and you can distinguish it from unformatted diskettes, BASIC diskettes, or diskettes for use with other systems.

While you are at it, repeat this formatting process on all the new diskettes that you want to use with the Apple Pascal System. With each new diskette, place it in the disk drive, type 4 and press the RETURN key.

You may wonder why your one-and-only disk drive is called "4". There's no good reason for this, it's just that the disk drive was assigned the number 4. Why, in Spanish, is the word for window "ventana"? It just happened that way.

When you have finished formatting all your new diskettes, and have written the word "Pascal" on each of them, answer the question

```
          FORMAT WHICH DISK (4, 5, 9..12) ?
```

with a simple press of the key marked RETURN . You get the message

```
          PUT SYSTEM DISK IN #4 AND PRESS RETURN
```

By "SYSTEM DISK" the Apple means "APPLEØ:" (unless you stopped after Step One of the startup procedure, and continued to use APPLE1: as your system disk). By "#4" the Apple means the disk drive. Sometimes your disk drive is called "DRIVE 4" and sometimes "#4:", but it's all the same thing.

Do as it says, place the diskette marked APPLEØ: in the disk drive (or, as we say in Apple Pascal jargon, "Put APPLEØ: in #4:") and press the RETURN key.

The Apple says:

```
          THAT'S ALL FOLKS...
```

And if you watch the top of the screen, the line:

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(INK, X(ECUTE, A(SSEM, D(EBUG,?
```

appears (of course, it doesn't all appear; but you know it's there, and can check with CTRL-A).

MAKING THE ACTUAL COPIES

As you have seen, you can get into the Filer by typing F when you have the COMMAND prompt line on the screen. You must have diskette APPLE1: or diskette APPLEØ: in the disk drive when you type F for the Filer, or (if APPLEØ: is your system diskette) you will get the message

```
          NO FILE APPLEØ:SYSTEM.FILER
```

If this happens, just put APPLEØ: in the disk drive and type F again.

The Filer is that portion of the system that allows you to manipulate information on diskettes. One of the Filer's abilities is to transfer information from one diskette to another. To invoke this facility, once you have the FILER prompt line on the screen, type T for T(ransfer.

This is what you see:

```
TRANSFER ?
```

Place diskette APPLE3: into the disk drive and answer the question as follows:

```
APPLE3:
```

which means that you want to transfer the entire contents of the source diskette called APPLE3: . After you have specified which diskette's information you want transferred (and pressed the key marked RETURN), the computer checks to make sure the correct diskette is in the disk drive. If you have forgotten to put diskette APPLE3: in the drive, then you will see the message

```
APPLE3:
NO SUCH VOL ON-LINE <SOURCE>
```

In that case you must type T for Transfer again, and repeat the process. With the correct source diskette in the drive, the Transfer process continues and the computer asks the next obvious question: If you are going to transfer something, then

```
TO WHERE ?
```

Answer this question by typing

```
BLANK:
```

This is the name of the destination diskette, onto which you want APPLE3:'s information transferred. "BLANK:" is any of the diskettes that you just formatted. When a diskette is formatted it is automatically given the name BLANK: . Incidentally, those colons (:) are very important. You use them to indicate that you are referring to an entire diskette, and not just a part of one.

After you have told the computer where you want APPLE3:'s information transferred (and pressed the key marked RETURN), it says:

```
TRANSFER 28Ø BLOCKS ? (Y/N)
```

This message is mainly there to give you a chance to abandon the transfer if you made a typing error in the names of the source or the destination diskettes. The phrase "28Ø BLOCKS" means merely "THE WHOLE DISKETTE". In any case, you type

```
Y
```

The disk whirrs and zicks a few times, and you see the message:

```
PUT IN BLANK:
TYPE <SPACE> TO CONTINUE
```

Do as it says. By the colon, you know that it means to put the diskette called BLANK: into the disk drive. The second line tells you to press the space bar when the diskette is in place (and the door closed, of course).

All the information which is on diskette APPLE3:, including the diskette's name, will be copied onto diskette BLANK:, completely overwriting BLANK:. Therefore, the computer warns you that you are about to lose any information that might be stored on BLANK:. It says

```
DESTROY BLANK: ?
```

Since you want to turn BLANK: into a perfect copy of APPLE3:, the answer is

```
Y
```

The process is under way. The computer will tell you to first put in one diskette and then the other. Follow the instructions. Your screen will look like this after a while:

```
PUT APPLE3: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT BLANK: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT APPLE3: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT BLANK: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT APPLE3: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT BLANK: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT APPLE3: IN UNIT #4
TYPE <SPACE> TO CONTINUE
PUT BLANK: IN UNIT #4
TYPE <SPACE> TO CONTINUE
```

and so on. You will have to insert the two diskettes a total of 2Ø times, and press the spacebar 2Ø times, to copy the entire diskette. When copying is done, the screen celebrates by saying

```
APPLE3:                      --> BLANK:
```

By this cryptic remark, the computer is telling you that the contents of APPLE3: , including the diskette's name, have been copied onto the diskette that used to be called BLANK: . This is just what you wanted. Now, writing lightly with a marking pen (do not use a pencil or a ballpoint pen), write "APPLE3:" on the new diskette's label. It is very important to label diskettes immediately, so you know what information they contain.

DO IT AGAIN, SAM

You should, at this time, make sure that you have at least one backup copy of each of the Pascal system diskettes: APPLEØ:, APPLE1:, APPLE2:, and APPLE3: . Then you should store the original diskettes away in a safe place.

When you are through making backup copies, be sure to put APPLEØ: (or APPLE1: if you are using that as your system diskette) back into the disk drive, BEFORE typing Q to Quit the Filer. If you forget to do this, the system will stop responding to the keyboard after you type Q ; you will have to turn the Apple off and repeat the entire startup procedure.

USING THE SYSTEM

A DEMONSTRATION

At last, the reward for all your work to this point: you are finally ready to use the Apple Pascal system to run a program. Diskette APPLE3: contains several small "demonstration" programs. To see a list of those programs, put APPLEØ: in the disk drive and enter the Filer (by typing F in response to the COMMAND prompt line, remember?). When the FILER prompt line appears on the screen, put APPLE3: in the drive and type L to List the diskette's directory. The Filer says:

```
DIR LISTING OF ?
```

In response, type the name of the diskette whose directory you wish to see:

```
APPLE3:
```

When you press the RETURN key, a long list of program files appears on the screen, many of them both in their .TEXT versions (the form in which they are written and edited) and also in their compiled .CODE versions (the form in which they can be executed). When the screen is full, the display stops and the message

```
TYPE <SPACE> TO CONTINUE
```

appears at the top of the screen. Press the Apple's spacebar to see the remaining files. For now, we are interested in the file named GRAFDEMO.CODE . But before executing this program, you must Transfer it to your system diskette, APPLEØ: (most graphics programs must use routines from the "system library", a file on APPLEØ: and also on APPLE1:). In response to the FILER prompt line, type

```
T
```

The Filer says

```
TRANSFER ?
```

Answer the question as follows:

```
APPLE3:GRAFDEMO.CODE
```

which means you want to transfer only the file named GRAFDEMO.CODE from the source diskette named APPLE3: . The Filer checks to see that APPLE3: is in the disk drive, and that it contains a file named GRAFDEMO.CODE, and then asks

```
TO WHERE ?
```

You know that you want a copy of the file GRAFDEMO.CODE transferred to the destination diskette APPLEØ: . To avoid confusion, let's give this copied file the same name when it is transferred to APPLEØ: . To do this, answer the question by typing

```
APPLEØ:GRAFDEMO.CODE
```

Note: you MUST specify a name for the file on the destination diskette. If you forget to type a file name, the Filer thinks you are referring to the entire diskette, and asks

```
DESTROY APPLEØ: ?
```

Since you do not wish to destroy APPLEØ: , type

```
N
```

Now, if you have typed all of your responses correctly, a new display appears:

```
PUT IN APPLEØ:
TYPE <SPACE> TO CONTINUE
```

Follow the directions, putting APPLEØ: in the disk drive and pressing the Apple's spacebar. You are soon rewarded with the message

```
APPLE3:GRAFDEMO.CODE
 --> APPLEØ:GRAFDEMO.CODE
```

This tells you that a copy of the file GRAFDEMO.CODE on diskette APPLE3: has been successfully transferred to a file named GRAFDEMO.CODE on diskette APPLEØ: . Since the system diskette APPLEØ: is already in the disk drive, you may now safely type Q to Quit the Filer. When the COMMAND prompt line appears, type X for X(ecute. The Apple says

```
EXECUTE WHAT FILE?
```

Answer by typing the name of the file you just transferred to APPLEØ:

```
          APPLEØ:GRAFDEMO
```

Note: DO NOT type the suffix .CODE ; the system knows you can execute only a code file, so it automatically supplies the suffix .CODE for you, in addition to any name that you type.

When this message appears:

```
          PRESS ANY KEY TO QUIT.
          PLEASE WAIT WHILE CREATING BUTTERFLY
```

the program is running. After a short pause, the display begins. Just sit back and enjoy it: soon you'll be writing your own programs yourself. When you are tired of watching, press the spacebar on the Apple's keyboard to return to the COMMAND prompt line. You can use this same procedure to run any of the programs on APPLE3: . These programs and their purposes are described in the Appendix A.

DO IT YOURSELF

Now, for some more experience at using the Apple Pascal system, let's try writing a little program. This discussion will assume that you are using your new copy of APPLEØ: as your system diskette (or "boot diskette" as it is often called). This copy is not write-protected and you have never used the Editor to create any new files on it before (it's all right if you have added the file GRAFDEMO.CODE to it).

With the COMMAND prompt line showing, and with APPLEØ: in the disk drive, type E to select the E(dit option. Soon, this message appears:

```
>EDIT:
NO WORKFILE IS PRESENT. FILE? ( <RET> FOR NO FILE <ESC-RET> TO EXIT )
:
```

As usual, you must use CTRL-A to see the right half of the message. This message gives you some information and some choices. The first word, >EDIT: , tells you that you are now in the Editor. The next sentence, NO WORKFILE IS PRESENT , tells you that you have not yet used the Editor to create a "workfile", which is a "scratchpad" diskette copy of a program you are working on. If there had been a workfile on APPLEØ: , that file would have been read into the Editor automatically.

Since there was no workfile to read in, the Editor asks you, FILE? If you now typed the name of a .TEXT file stored on APPLEØ:, that textfile would be read into the Editor. However, there are no .TEXT files on APPLEØ: yet, and besides, you want to write a new program. In parentheses, you are shown how to say that you don't want to read in an old file: <RET> FOR NO FILE . This means that, if you press the Apple's RETURN key, no file will be read in and you can start a new file of your own. That's just what you want to do, so press the Apple's RETURN key

(the rest of the message says if you first press the ESC key and THEN press the RETURN key, you'll be sent back to the COMMAND prompt line). When you have pressed only the RETURN key, the full EDIT prompt line appears:

```
>EDIT: A(DJST C(PY D(LETE F(IND I(NSRT J(MP R(PLACE Q(UIT X(CHNG Z(AP
```

The chapter called THE EDITOR in the Apple Pascal Operating System Manual explains all of these command options in detail; for now you will only need a few of them. The first one you will use is I(NSRT , which selects the Editor's mode for inserting new text. Type I to select Insert mode, and this prompt line appears:

```
>INSERT: TEXT [<BS> A CHAR,<DEL> A LINE] [<ETX> ACCEPTS,<ESC> ESCAPES]
```

As long as this line is showing at the top of the screen, anything you type will be placed on the screen, just to the left of the white square "cursor". If the cursor is in the middle of a line, the rest of the line is pushed over to make room for the new text. If you make a mistake, just use the left-arrow key to backspace over the error, and then retype. At any time during an insertion, if you press the Apple's ESC key your insertion will be erased. At any time during an insertion, if you press CTRL-C the insertion will be made a permanent part of your file, safe from being erased by ESC or by the left-arrow key. You can then type I to reenter Insert mode and type more text.

Now for our program. With the INSERT prompt line showing, press the RETURN key a couple of times, to move the cursor down, and then type

```
          PRORAFM DEMO;
```

You can use any name for your program, but in this discussion it will be called DEMO . Now press CTRL-C (type C while holding down the CTRL key). Your insertion so far is made "permanent", and the EDIT prompt line reappears. But, horrors! You made several typing errors when typing the word PROGRAM . Since you have already pressed CTRL-C , it is too late to backspace over your errors and retype them.

Fortunately, there are other ways. First, let's correct the missing G in PROGRAM . Using the left arrow key, move the cursor left until it is sitting directly on the R . Then type I to reenter Insert mode. Ignore the fact that the remainder of the line seems to have suddenly disappeared, and type the missing letter G . When you press CTRL-C to make this insertion permanent, the rest of the line returns:

```
          PROGRAFM DEMO;
```

The letter F is certainly not needed, so move the cursor right (using the right-arrow key) until it is sitting directly on the F . Now type D to select the Editor's D(LETE option. When the DELETE prompt line appears, press the right-arrow key once. The offending F instantly disappears. What happens next is similar to Insert mode: if you press the ESC key, the deletion is forgotten, as if it had never happened. If you press CTRL-C , the deletion is made a permanent part of your file. To remove that F permanently, press CTRL-C . The line closes in to fill the deleted letter's place:

```
          PROGRAM DEMO;
```

Now you know how to use the Editor's Insert and Delete modes to write text and to correct your errors. Try typing the rest of program DEMO into your file. Be sure to "accept" your insertions, from time to time, by pressing CTRL-C . That way, you minimize your loss if you accidentally press the ESC key. Here is the complete program:

```
          PROGRAM DEMO;

          USES TURTLEGRAPHICS, APPLESTUFF;
          VAR ANGLE, DISTANCE : INTEGER;

          PROCEDURE CRAWL;
          BEGIN
            MOVE (2 * DISTANCE);
            TURN (ANGLE)
          END;

          BEGIN
            ANGLE := Ø;
            REPEAT
              INITTURTLE;
              PENCOLOR (WHITE);
              FOR DISTANCE := 1 TO 99 DO CRAWL;
              ANGLE := ANGLE + 5
            UNTIL KEYPRESS;
            TEXTMODE
          END.
```

When you are typing this program, the punctuation and spelling must be exactly as shown. The indentation of the lines is not important, but it easier to read as shown. You will notice that, once you have started a new indentation, the Editor maintains that indentation for you. To move back to the left, just press the left-arrow key before you type anything on the new line.

Program DEMO makes use of graphics routines in the Unit TURTLEGRAPHICS, and uses the keypress function from the Unit APPLESTUFF (see Chapter 7 for more details). The third line of the program declares two integer variables, DISTANCE and ANGLE. Next, a Pascal procedure named CRAWL is defined, between the first BEGIN and END; . From here on, each time this new Pascal statement CRAWL is used, a graphics "turtle" will trace a line on the screen, of length 2*DISTANCE moving in the current direction, and will then change the direction by an amount ANGLE.

The next BEGIN and the last END. outline the main program. The portion of the program from REPEAT to UNTIL KEYPRESS is repeated over and over again, until any key on the Apple's keyboard is pressed.

In each repetition, the screen is cleared and the tracing color is set to WHITE. Then the procedure CRAWL is performed, first with the value of DISTANCE set to one, then with DISTANCE set to the value two, and so on, until DISTANCE is set to 99 . The "turtle" moves, then turns, then moves some more, then turns again, and so on, for 99 steps. That completes one design on the screen. In the next repetition, if no key has been pressed, the ANGLE has increased by 5 degrees, the screen is cleared by INITTURTLE, and the whole process starts again.

Now you should save this program. With the EDIT prompt line showing, type Q to select the Q(UIT option. The following message appears:

```
          >QUIT:
               U(PDATE THE WORKFILE AND LEAVE
               E(XIT WITHOUT UPDATING
               R(ETURN TO THE EDITOR WITHOUT UPDATING
               W(RITE TO A FILE NAME AND RETURN
```

Type U to create a "workfile" diskette copy of your program (future versions of this file will be "Updates"). This workfile is a file on your boot diskette called SYSTEM.WRK.TEXT . The Apple says

```
          WRITING...
          YOUR FILE IS 33Ø BYTES LONG.
```

(the number of bytes may be a little different) and then the COMMAND prompt line reappears. Now type R to select the R(UN option. This automatically calls the Compiler for you, since the workfile contains text. If you have typed the program perfectly, the following messages (again, perhaps with slightly different numbers) appear, one by one:

```
          COMPILING...


          PASCAL COMPILER II.1 [B2B]
          <   Ø>....
          TURTLEGR [ 2483 WORDS]
          <   5>..........................
          APPLESTU [ 1Ø78 WORDS]
          <  3Ø>..................
          CRAWL    [ 1Ø98 WORDS]
          <  46>.....
          DEMO     [ 11Ø9 WORDS]
          <  51>........
          59 LINES
          SMALLEST AVAILABLE SPACE = 1Ø98 WORDS
```

If the Compiler discovers mistakes, it will give you a message such as

```
          PROFRAM <<<<
          LINE 2, ERROR 18: <SP>(CONTINUE), <ESC>(TERMINATE), E(DIT
```

Don't despair; just type E for E(DIT . Your workfile will be automatically read back into the Editor for repairs. Read the error message at the top of the screen, press the spacebar, and make any necessary changes using I(nsert and D(elete. Then Q(uit, U(pdate the workfile, and R(un your program again, by typing Q U R (the Apple will store up several commands in advance).

When your program has been successfully Compiled, it is automatically executed. You will see the message

```
          RUNNING...
```

and then a horizontal line appears on the screen. That is the first design your program draws: the white "turtle" moves out a distance 2*1 , turns an angle Ø ; moves 2*2 , turns Ø ; moves 2*3 , turns Ø ; etc. Keep watching as successive designs turn through larger and larger angles between moves. When you want to interrupt the program, press any key on the keyboard.

Try making changes to the program, by setting a different starting ANGLE, or a different increment to the ANGLE, or a different distance to MOVE. To do this, type E for E(DIT, use I(NSRT and D(LETE to make changes, and then Q(uit, U(pdate the workfile, and R(un again by typing Q U R . This cycle of Edit-Run-Edit-Run is the basis of all program development in the Apple Pascal system.

The workfile on APPLEØ: now contains the text version of your program in a file named SYSTEM.WRK.TEXT , and the compiled P-code version of your program in another file named SYSTEM.WRK.CODE . When your program is running as you want it to, you should save the text and code workfile under other filenames. With the COMMAND prompt line showing, type F to enter the Filer. When the FILER prompt line appears, type S for S(ave. You will be asked

```
          SAVE AS ?
```

and you should respond by typing any filename with fewer than 1Ø characters. For example, you might type

```
          DEMO
```

This changes the names of the workfile from SYSTEM.WRK.TEXT to DEMO.TEXT , and from SYSTEM.WRK.CODE to DEMO.CODE . If you want to keep a permanent copy of your program on another diskette, you should now use the T(ransfer command to transfer DEMO.TEXT and DEMO.CODE, one at a time, to the other diskette. Remember to wait for the prompt message before removing the source diskette from the drive and putting in the destination diskette.

WHAT TO LEAVE IN THE DRIVE

When you turn the Apple off, it is a good idea to leave the diskette called APPLE1: in the disk drive. If some other diskette or no diskette is in the drive when the Apple is turned on, the drive will spin indefinitely. If this continues for hours and hours, some wear will take place on the drive and any diskette in it. So, it is a good idea to make a habit of leaving a copy of APPLE1: (now that you have copies) in the disk drive when you turn the system off. (APPLEØ: will not do, as it is missing a file that is needed for the first stage of system startup.)

Of course, if you turn on the system and APPLE1: is not in the drive, just press the key marked RESET . Place APPLE1: in the drive and turn the system off and then on again. No damage results from turning on the Apple with the wrong diskette (or no diskette) in the drive. Gradual, unnecessary wear results from leaving the disk drive running for a long period of time with the incorrect diskette (or no diskette) in the drive.

ONE-DRIVE SUMMARY

STARTING UP THE SYSTEM

To start the system, place diskette APPLE1: in the disk drive; then turn on the Apple's power. When the "WELCOME" message appears, Pascal is running. Using APPLE1: as the system diskette, you can file, edit, and execute previously compiled programs; but you cannot compile new programs. To change system diskettes, place APPLEØ: in the drive; then press the Apple's RESET key. Again, when the "WELCOME" message appears, Pascal is running. Using APPLEØ: as the system diskette, you can file, edit, compile, and execute programs; but you cannot start up the system from power-on.

FORMATTING NEW DISKETTES

```
To format a diskette, have Pascal's COMMAND prompt line showing.
Place diskette APPLE3: in the disk drive, and type
          X
In response to the query:
          EXECUTE WHAT FILE?
type
          APPLE3:FORMATTER
When the question:
          FORMAT WHICH DISK ?
appears, place the new diskette in the disk drive, then type
          4
```

and press the RETURN key. The diskette will be formatted. To leave the formatting program, press the RETURN key in response to the WHICH DISK question. A newly formatted diskette has the name BLANK:

COPYING DISKETTES

To copy a diskette, have the COMMAND prompt line showing, and put diskette APPLEØ: or APPLE1: in the disk drive. Get into the Filer by typing

F

When the FILER prompt line appears, put into the disk drive the source diskette to be copied. Then type

T

To the question:

TRANSFER ?

reply by typing the name of the source diskette to be copied, and then press the RETURN key. For example:

APPLE3:

To the next question:

TO WHERE ?

reply with the name of the destination diskette that is to become the backup copy. For example:

BLANK:

Then follow the instructions displayed on the screen, switching the diskettes back and forth until the copy is complete. Before you Quit the Filer, be sure to put your system diskette (usually APPLEØ:) back in the drive.

Note: you cannot make a copy onto a destination diskette that has the same name as the source diskette. Use the Filer to C(hange the name of either diskette, at least while making the copy.

EXECUTING A PROGRAM

To execute a previously compiled program, put your system diskette (APPLEØ: or APPLE1:) into the disk drive. With the COMMAND prompt line showing, enter the Filer by typing

F

When the FILER prompt line appears, put into the disk drive the diskette containing the program codefile that you wish to execute. Then type

T

for T(ransfer. To the question

TRANSFER ?

reply by typing the name of the program's diskette and codefile. For example,

APPLE3:GRAFDEMO.CODE

To the next question

TO WHERE ?

reply with the name of your system diskette, and the same filename (or another name, if you wish). For example,

APPLEØ:GRAFDEMO.CODE

When you are prompted

PUT IN APPLEØ:

follow the instructions, and press the spacebar. The program is then transferred onto your system diskette, which is where it must be in order to execute it. Now type Q to Q(uit the Filer, and when the COMMAND prompt appears, type X for X(ecute. When the Apple prompts

EXECUTE WHAT FILE?

answer by typing the name of your system diskette and the newly transferred codefile you wish to have executed. DO NOT type the .CODE suffix. In this example, you would type

APPLEØ:GRAFDEMO

The program should now run.

WRITING A PROGRAM

To start a new file in the Editor, put your system diskette (which must be APPLEØ: if you want to R(un your program) into the disk drive. With the COMMAND prompt line showing, type F to enter the Filer. Then type N for N(ew. If you are asked

THROW AWAY CURRENT WORKFILE ?

type Y for Y(es. When you see the message

WORKFILE CLEARED

type Q to Q(uit the Filer, and then type E to enter the Editor. This message appears:

```
>EDIT:
NO WORKFILE IS PRESENT. FILE? ( <RET> FOR NO FILE <ESC-RET> TO EXIT )
```

Press the RETURN key, and the full EDIT: prompt line appears. You can now insert text at the cursor position by typing I for I(nsert and then typing your program. Conclude each insertion by pressing CTRL-C. Delete text at the cursor position by typing D for D(elete and then moving the cursor to erase text. Conclude each deletion by pressing CTRL-C . When you have written a version of your program, type Q to Q(uit the Editor, and then type U to U(pdate the workfile to contain your latest program version.

With the COMMAND prompt line showing, you can then type R to R(un your program. This automatically compiles the text workfile (using the Compiler program on APPLEØ:), stores the compiled code workfile, and executes it. To reenter the Editor, type E in response to the COMMAND prompt. The text workfile is automatically read back into the computer.

When a version of your program is complete, you can U(pdate the text workfile to contain that latest version and R(un the program to create a code workfile of that version. To save the workfile versions of your program on another diskette for later use, first save the workfile under another name on your system diskette (APPLEØ:). Type F in response to the COMMAND prompt to enter the Filer. Then type S for S(ave. When you see the prompt

SAVE AS ?

type the name of your system diskette and the filename under which you want your program saved. Do not type any .TEXT or .CODE suffix. For example, if you want your program saved under the filename DEMO , you might type

APPLEØ:DEMO

The text workfile SYSTEM.WRK.TEXT is saved as DEMO.TEXT on APPLEØ:, and the code workfile SYSTEM.WRK.CODE is saved as DEMO.CODE .

Now you can T(ransfer those files to any other diskette, for safe keeping. Type

T

and when the Filer asks

TRANSFER ?

give the name of one of the S(aved files on your system diskette. In the previous example, you could type APPLEØ:DEMO.TEXT To the next question TO WHERE ? reply by typing the name of the diskette and file where you wish your program file to be stored. For example, you might type MYDISK:DEMO.TEXT The Apple will prompt you when it is time to put the destination diskette in the drive. When the text version of your program has been transferred onto the destination diskette, put your system diskette back in the drive. Now, type T for T(ransfer again, and transfer the code version of your program to the destination diskette in the same way you transferred the text version.

Remember to put APPLEØ: back in the disk drive before Q(uitting the Filer.

APPENDIX E
TWO-DRIVE STARTUP

This appendix is a tutorial session to get you started using the Language System with Pascal, on an Apple II with two or more diskette drives. If your system has only one diskette drive, please go back and read Appendix D instead.

EQUIPMENT YOU WILL NEED

You should have the following:

1. Your 48K Apple computer, with a Language Card installed, and at least two disk drives. The first two should be attached to a disk controller card in slot 6. All your disk controller cards should have the new PROMs, P5A and P6A, that came with the Language System.

2. A TV set or video monitor, connected to your Apple.

3. The following Language System diskettes:

   a. APPLE1:
   b. APPLE2:
   c. APPLE3:
   d. A blank diskette
   e. A second blank diskette

The diskette marked "APPLE1:" is needed to start the system. The diskette marked "APPLE2:" adds certain extra features to the system (the Compiler, the Assembler, and the Linker). You will not need the diskette marked "APPLE2:" until later. The diskette marked "APPLE3:" contains a number of useful utility programs. A diskette marked "APPLEØ:" is also included with the Language System. This diskette is normally used with single-drive systems.

The Apple and the TV or monitor should be plugged in. Turn on the TV now, so that it can warm up; but leave the Apple turned off.

MORE THAN TWO DISK DRIVES

If your system has more than two disk drives, the third drive gets connected to the "DRIVE 1" pins on the second controller, which goes in slot 5. A fourth drive is connected to the "DRIVE 2" pins on the second controller, in slot 5. A fifth and even a sixth drive can be connected to a controller in slot 4, using the "DRIVE 1" and "DRIVE 2" pins, respectively.

NUMBERING THE DISK DRIVES

Pascal assigns a "volume" number to each of the disk drives. It is not a bad idea to place tags with these numbers on your disk drives. Here's how the volume numbers are assigned to the various disk drives:

APPLE DISK DRIVE	PASCAL VOLUME
Slot 6, Drive 1	#4:
Slot 6, Drive 2	#5:
Slot 5, Drive 1	#11:
Slot 5, Drive 2	#12:
Slot 4, Drive 1	#9:
Slot 4, Drive 2	#1Ø:

You will find that you can refer to any diskette by either the name of the diskette (e.g., APPLE3:) or by the volume number of the drive in which it sits (e.g., #11:)

PASCAL IN SECONDS

Place the diskette marked "APPLE1:" in disk drive #4: (slot 6, drive 1). If you are not familiar with handling diskettes, see the manuals that came with your disk drives. Diskettes must be treated correctly if they are to last.

Close the door to disk drive #4: , and turn on the Apple. The rest is automatic. First, the message

```
APPLE II
```

appears at the top of your TV or monitor screen, and disk drive #4:'s "IN USE" light comes on. The disk drive emits a whirring, zickking sound that is as pleasant as a cat's purring, since it lets you know that everything is working. The screen lights up for an instant with a display of black at-signs (@) on a white background, then goes black again. Next the other disk drives are turned on, one at a time, as Apple Pascal finds out what is in each drive. A drive with no diskette in it may buzz and clatter a bit. When Apple Pascal cannot read anything from a disk drive, it recalibrates the drive's read-head position (buzz, clatter) and then tries again. Now disk drive #4: stops entirely for a moment; then it whirrs some more. Finally, the message

```
WELCOME  APPLE1,  TO
U.C.S.D.  PASCAL SYSTEM II.1
CURRENT DATE IS 26-JUL-79
```

appears (the date will be different), followed in a second or so by a line at the top of the screen:

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(IN
```

This line at the top of the screen is called a "prompt line". When you see this prompt line, you know that your Apple computer is running the Apple Pascal system.

Starting the system depends only on having APPLE1: in disk drive #4:. This time, you left the other drives empty; but you will soon discover that the system starts more quickly and quietly if the other drives have Pascal diskettes in them. For now, you could put diskettes APPLE2: and APPLE3: in any empty disk drives. Later, you will have other diskettes to put in them. In any case, make sure you never put two diskettes with the same name into the system at the same time. This may cause the directories of those diskettes to get scrambled.

CHANGING THE DATE

The date that comes on the diskette will not be correct. It is a good habit to reset the date the first time you use the Pascal System on any given day. It only takes a few seconds. Press F on the keyboard (without pressing the RETURN key or any other keys). The screen goes blank, and then this line appears at the top:

```
FILER: G, S, N, L, R, C, T, D, Q
```

This is a new prompt line. Prompt lines are named after their first word. The prompt line you first saw was the "COMMAND" prompt line. This one is the "FILER" prompt line. Sometimes we say that you are "in the Filer" when this line is at the top of the screen. Each of the letters on the prompt line represents a task that you can ask the Apple to do. For example, to change the date, press D (again, just type the single key, without pressing RETURN or any other keys).

When you do, another message is put on the screen. It says:

```
DATE SET: <1..31>-<JAN..DEC>-<ØØ..99>
TODAY IS 26-JUL-79
NEW DATE ?
```

It doesn't really mean that today is 26-JUL-79 (or whatever date your screen shows), but that the Apple THINKS that is today's date. Since it isn't, you can change the date to be correct. The correct form for typing the date is shown on the second line of the message: one or two digits giving the day of the month, followed by a minus sign, followed by the first three letters of the name of the month, followed by another minus sign, followed by the last two digits of the current year. Then press the key marked RETURN .

If the month and year are correct (as they will often be, when you change the date) all you have to do is type the correct day of the month, and press the RETURN key. The system will assume that you mean to keep the same month and year displayed by the message. If you type a day and a month, the system will assume you mean to keep only the year the same.

Go ahead and make the date correct. This is your first interaction with the system, and is typical of how the system is used. In general, at the top of the screen there will usually be a prompt line which represents several choices of action. When you type the first letter of one of the choices, either you will be shown a new prompt line giving a further list of choices, or else the system will carry out the desired action directly. If you type a letter that does not correspond to one of the choices, the prompt line blinks but otherwise nothing happens. Remember to type only a single letter to indicate your choice; it is not necessary to press the RETURN key afterward.

Sometimes, as when setting the date, you are asked to type a response of several characters. You tell the system that your response is complete by pressing the RETURN key. If you make a typing error before pressing the RETURN key, you can back up and correct the error by pressing the left-arrow key. You should experiment by making deliberate errors in entering a date, and then erasing the errors with the left-arrow key.

One further note. Normally, your new date is saved on diskette APPLE1:, so the system "remembers" this date the next time you turn the Apple on. However, since you are using the write-protected diskettes that came with your Language System, your new date was not permanently saved. The next time you turn the Apple off, the new date will be "forgotten". By the end of this session, you will have made backup copies of the Language System diskettes. From then on, you will use these copies, which are not write-protected, and your date changes will be saved.

MAKING BACKUP DISKETTE COPIES

WHY WE MAKE BACKUPS

Ask yourself this question: What would happen to your system if you were to lose or damage one of the system diskettes (APPLE1:, APPLE2:, or APPLE3:)? It would be as bad as losing your Apple itself, as far as your being able to use Apple Pascal.

These diskettes are quite precious. The first thing you should do, therefore, is to make backup copies of them. Afterward, you should never use the originals, but put them someplace where the temperature is moderate, where there is no danger of them getting wet, and where such diskette destroyers as dogs, dirt, children, and magnetic fields cannot get at them.

A truly cautious person will keep on hand two backup copies of each original. That way, you will need to use an original only in the very rare case when both of its backup copies are lost (when one copy is lost or damaged, another backup copy is made from the surviving backup copy). If your backups were damaged or erased while in use, find out why they were destroyed before inserting your only surviving copy. Using diskettes for which you have backups, repeat the procedure that destroyed the first diskettes, and if you can't figure out what the problem is, bring your system to the dealer to make sure it is working correctly.

HOW WE MAKE BACKUPS

The Pascal system can copy all the information from one diskette (or any portion of the information) onto another diskette. But the system cannot store information on a new diskette, just as that diskette comes from the computer store. Therefore, the system is supplied with a program that allows you to take any 5-inch floppy diskette and "format" it so that it will work with the Apple Pascal system.

Incidentally, this is one of the nice little things about the Apple system: ANY high-quality 5-inch floppy diskette (Apple recommends diskettes made by Dysan Corporation) will work on it. Some systems require you to have "1Ø sector" or "15 sector" or "soft sectored" diskettes. The Apple doesn't care, it takes any of these kinds of diskettes, and (through the FORMATTER program) makes them into the kind of diskette it needs.

If you have been following this session by carrying out the instructions on your Apple, the FILER prompt line should be showing at the top of the screen:

```
FILER: G, S, N, L, R, C, T, D, Q
```

Type Q on the keyboard to Quit the Filer.

GETTING THE BIG PICTURE

When you Quit the Filer, disk drive #4: whirrs, and you see the COMMAND prompt line again:

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(IN
```

There is actually more of this prompt line, off to the right of your TV or monitor. To see the rest of the screen, hold down the key marked CTRL and, while holding it, press the "A" right alongside it. (Or, to be brief, we say: "press CTRL-A".)

You now see

```
K, X(ECUTE, A(SSEM, D(EBUG,?
```

This is simply the rest of the line that began "COMMAND:". All together, the full prompt line would look like this:

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(INK, X(ECUTE, A(SSEM, D(EBUG,?
```

The Apple Pascal system displays information on a "screen" that is 8Ø characters wide, but your TV or monitor shows only the leftmost 4Ø characters or the rightmost 4Ø characters at any one time. You use the CTRL-A trick whenever you wish to see if there is more stuff on the other "half" of the screen. Repeated pressing of CTRL-A flips back and forth between the left half of the screen and the right half.

Also, sometimes the TV display will seem to be blank. This might mean that you are just staring at the empty right half of the screen. Before you come to the conclusion that something is wrong, always try CTRL-A. You get back to the left side of the screen by typing CTRL-A again, and you might find that everything is OK after all.

Summary of this diversion: The screen is really twice as wide as it looks. To flip from the left side to the right side or back again, you type CTRL-A.

FORMATTING NEW DISKETTES

Place diskette APPLE3: in any available disk drive except drive #4: . This has to be done because the FORMATTER program is on APPLE3:. Now, with the COMMAND prompt line at the top of the screen, type

```
X
```

and the screen responds:

```
EXECUTE WHAT FILE?
```

You type

```
APPLE3:FORMATTER
```

and press the key marked RETURN .

The disk drive containing APPLE3: whirrs a bit and the screen says:

```
APPLE DISK FORMATTER PROGRAM
FORMAT WHICH DISK (4, 5, 9..12) ?
```

Take all the new, blank diskettes that you are going to use with the Pascal System (but not, of course, any diskettes that have precious information on them, such as the diskettes that came with the Pascal System) and place them in a pile. Their labels should be blank. Make sure that you don't have any diskettes with data in a non-Pascal format, such as BASIC diskettes: the Pascal system will be unable to read them, and will regard them as blank, erasing any old information in the formatting process.

Remove the diskette in disk drive #5: (if yours is a two-drive system, you will be removing diskette APPLE3:) and put one of the new, blank diskettes into that drive. Then type

```
5
```

and press the key marked RETURN .

If the diskette in drive #5: has already been formatted, you will receive a warning. For example, if you have left APPLE3: in that drive you will be warned with the message

```
DESTROY DIRECTORY OF APPLE3 ?
```

At this point you can type

```
N
```

(which stands for "No") without pressing the RETURN key, and your diskette will not be destroyed. Let's assume that you have a new, unformatted diskette. Then you will not get any warning, but the Apple will place this message on the screen:

```
NOW FORMATTING DISKETTE IN DRIVE 5
```

Disk drive #5: will make some clickings and buzzings and begin to whirr and zick. The process takes about 32 seconds. When formatting is complete, the screen again shows the message

```
FORMAT WHICH DISK (4, 5, 9..12) ?
```

Now you have a formatted diskette. We suggest that you write "Pascal" in small letters at the top of the diskette's label, using a marking pen. Do not use a pencil or ballpoint pen, as the pressure of writing may damage the diskette. The label will let you know that the diskette is formatted for use with the Apple Pascal system, and you can distinguish it from unformatted diskettes, BASIC diskettes, or diskettes for use with other systems.

While you are at it, repeat this formatting process on all the new diskettes that you want to use with the Apple Pascal System. With each new diskette, place it in drive #5: , type 5 and press the RETURN key.

Note: If you have more than two drives, you can simplify the procedure by putting the next diskette to be formatted into any unoccupied drive. Then, when the system asks

```
FORMAT WHICH DISK (4, 5, 9..12) ?
```

just type the correct volume number of the drive containing your new, blank diskette, and then press the RETURN key. This will save you some diskette-swapping.

When you have finished formatting all your new diskettes, and have written the word "Pascal" on each of them, answer the question

```
FORMAT WHICH DISK (4, 5, 9..12) ?
```

with a simple press of the key marked RETURN . You get the message

```
THAT'S ALL FOLKS...
```

And if you watch the top of the screen, the line

```
COMMAND: E(DIT, R(UN, F(ILE, C(OMP, L(INK, X(ECUTE, A(SSEM, D(EBUG,?
```

appears (of course, it doesn't all appear; but you know it's there, and can check with CTRL-A).

MAKING THE ACTUAL COPIES

As you have seen, you can get into the Filer by typing F when you have the COMMAND prompt line on the screen. You must have diskette APPLE1: or diskette APPLE∅: in one of the disk drives when you type F to enter the Filer. If you forget (and APPLE1: is your system diskette), you will get the message

```
NO FILE APPLE1:SYSTEM.FILER
```

If this happens, just put APPLE1: in any drive and type F again.

The Filer is that portion of the system which allows you to manipulate information on diskettes. One of the Filer's abilities is to transfer information from one diskette to another. To invoke this facility, once you have the FILER prompt line on the screen, type T for T(ransfer.

This is what you see:

```
TRANSFER ?
```

Let's say that you want to make a backup copy of diskette APPLE3: , by copying APPLE3: onto one of your newly formatted diskettes. Put APPLE3: into any available disk drive, and put a newly formatted diskette into any other drive. If your system has only two drives, you will have to remove diskette APPLE1: from drive #4: . Once the FILER prompt line is showing, APPLE1: is no longer needed until you wish to Quit the Filer and return to the COMMAND prompt line. Now, answer the question by typing the name of the source diskette to be copied:

```
          APPLE3:
```

When you press the RETURN key, the computer checks to see that diskette APPLE3: is in one of the disk drives. If it is not, you will see the message

```
          APPLE3:
          NO SUCH VOL ON-LINE <SOURCE>
```

In that case, just put APPLE3: in a disk drive and type T for Transfer again. If the computer succeeds in finding APPLE3:, it asks you the next obvious question: If you are going to transfer something, then

```
          TO WHERE ?
```

Answer this question by typing the name of the diskette that is to become an exact backup copy of APPLE3:

```
          BLANK:
```

Remember that BLANK: is the name given to all newly formatted diskettes by the FORMATTER program. The colons (:) that appear after the diskette names are quite significant: they indicate that the entire diskette is being referred to.

After you have told the computer where you want APPLE3:'s information transferred (and pressed the key marked "RETURN"), it checks to see that BLANK: is also in one of the disk drives. If it is not, you will see the message

```
          PUT IN BLANK:
          TYPE <SPACE> TO CONTINUE
```

In that case, put BLANK: into any disk drive except the one containing APPLE3:, and press the Apple's spacebar. When the computer succeeds in finding both the source and the destination diskettes, it says

```
          TRANSFER 28Ø BLOCKS ? (Y/N)
```

This message is mainly there to give you a chance to abandon the transfer if you made a typing error in the names of the source or the destination diskettes. The phrase "28Ø BLOCKS" means merely "THE WHOLE DISKETTE". In any case, you type

```
          Y
```

All the information on diskette APPLE3:, including the diskette's name, will be copied onto diskette BLANK:, completely overwriting BLANK:. Therefore, the computer warns you that you are about to lose any information that might be stored on BLANK:. It says

```
          DESTROY BLANK: ?
```

Since you want to turn BLANK: into a perfect copy of APPLE3:, the answer is

```
          Y
```

The process is under way. It takes about two minutes to copy and check the entire diskette. When copying is done the screen celebrates by saying:

```
          APPLE3:                        --> BLANK:
```

by which cryptic remark the computer is telling you that the contents of APPLE3:, including the diskette's name, have been copied onto the diskette that used to be called BLANK:. This is just what you wanted.

There are now two diskettes with the same name, both in the system at once. This is a risky situation, confusing both to you and to the computer, so be sure to remove the new copy right away. Now, using a marking pen, write "APPLE3:" on the new diskette's label. Do not use a pencil or a ballpoint pen, as the pressure of writing may damage the diskette. It is very important to label diskettes immediately, so you know what information is stored on them.

DO IT AGAIN, SAM

You should, at this time, make sure that you have at least one backup copy of each of your system diskettes: APPLE1:, APPLE2:, and APPLE3:. In each case, just place the source diskette to be copied from in one drive, the blank destination diskette to be copied onto in another drive, and then type T to begin the Transfer. While you are at it, make a backup copy of APPLEØ: , too. It may come in handy, later on.

BEFORE you type Q to Quit the Filer and return to the COMMAND prompt line, be sure to put diskette APPLE1: back into drive #4: If you forget to do this, the computer will stop responding to its keyboard after you type Q ; even the RESET key will have no effect. You will have to turn the computer off, put APPLE1: in drive #4:, and turn the computer on again.

Finally, you should store the original diskettes (and one extra copy, if you like to be really safe) away, in a safe place.

USING THE SYSTEM

A DEMONSTRATION

At last, a reward for all your work to this point: you are finally ready to use the Apple Pascal system to run a program. Diskette APPLE3: contains several "demonstration" programs. To see a list of those programs, put APPLE3: in any disk drive except #4: (APPLE1: must be in drive #4:). Now, enter the Filer by typing F in response to the COMMAND prompt line. When the FILER prompt line appears on the screen, type L to List a diskette's directory. The Filer says:

```
DIR LISTING OF ?
```

In response, type the name of the diskette whose directory you wish to see:

```
APPLE3:
```

A long list of program files now appears on the screen, many of them both in their .TEXT versions (the form in which they are written and edited) and also in their compiled .CODE versions (the form in which they can be executed). When the screen is full, the display stops and the message

```
TYPE <SPACE> TO CONTINUE
```

appears at the top of the screen. Press the Apple's spacebar to see the remaining files. For now, we are interested in the file named GRAFDEMO.CODE .

Since the system diskette APPLE1: is already in disk drive #4: , you may now type Q to Quit the Filer. When the COMMAND prompt line appears, type X for X(ecute. The computer says

```
EXECUTE WHAT FILE?
```

Answer by typing the name of the diskette and file you wish to have executed:

```
APPLE3:GRAFDEMO
```

Note: DO NOT type the suffix .CODE ; the system knows you can execute only a code file, so it automatically supplies the suffix .CODE for you, in addition to any name that you type.

When this message appears

```
PRESS ANY KEY TO QUIT.
PLEASE WAIT WHILE CREATING BUTTERFLY
```

the program is running. After a short pause, the display begins. Just sit back and enjoy it: soon you'll be writing your own programs using these and other features of Apple Pascal. When you are tired of watching, press the spacebar on the Apple's keyboard to return to the COMMAND prompt line. You can use this same procedure to run any of the programs on APPLE3: . These programs are discussed in Appendix A.

DO IT YOURSELF

Now, for some more experience at using the Apple Pascal system, let's try writing a short program. This discussion will assume that you are using your new copies of the Pascal diskettes. You should be using a new copy of APPLE1: as your system diskette (or "boot diskette" as it is often called). This copy is not write-protected, and you have never used the Editor to create any new files on it before. Put the new copy of APPLE1: in the boot drive, volume #4: . You should also put a copy of APPLE2: in any other drive (APPLE2: contains the Compiler program).

With the COMMAND prompt line showing, type E to select the E(dit option. Soon, this message appears:

```
>EDIT:
NO WORKFILE IS PRESENT. FILE?( <RET> FOR NO FILE <ESC-RET> TO EXIT )
:
```

As usual, you must use CTRL-A to see the right half of the message. This message gives you some information and some choices. The first word, >EDIT: , tells you that you are now in the Editor. The next sentence, NO WORKFILE IS PRESENT , tells you that you have not yet used the Editor to create a "workfile", which is a "scratchpad" diskette copy of a program you are working on. If there had been a workfile on APPLE1: , that file would have been read into the Editor automatically.

Since there was no workfile to read in, the Editor asks you, FILE? If you now typed the name (including the drive's volume number or the diskette's name) of a .TEXT file stored on APPLE1: or on APPLE2:, that textfile would be read into the Editor. However, there are no .TEXT files on APPLE1: or APPLE2: yet, and besides, you want to write a new program. In parentheses, you are shown how to say that you don't want to read in an old file: <RET> FOR NO FILE . This means that, if you press the Apple's RETURN key, no file will be read in and you can start a new file of your own. That's just what you want to do, so press the Apple's RETURN key (the rest of the message says if you first press the ESC key and THEN press the RETURN key, you'll be sent back to the

COMMAND prompt line). When you have pressed the RETURN key, the full EDIT prompt line appears:

```
>EDIT: A(DJST C(PY D(LETE F(IND I(NSRT ...
```

The chapter called THE EDITOR in the Apple Pascal Operating System Reference Manual explains all of these command options in detail; for now you will only need a few of them. The first one you will use is I(NSRT , which selects the Editor's mode for inserting new text. Type I to select Insert mode, and yet another prompt line appears:

```
>INSERT: TEXT [<BS> A CHAR,<DEL> A LINE] [<ETX>ACCEPTS, <ESC>ESCAPES]
```

As long as this line is showing at the top of the screen anything you type will be placed on the screen, just to the left of the white square "cursor". If the cursor is in the middle of a line, the rest of the line is pushed over to make room for the new text. If you make a mistake, just use the left-arrow key to backspace over the error, and then retype. At any time during an insertion, if you press the Apple's ESC key your insertion will be erased. At any time during an insertion, if you press CTRL-C the insertion will be made a permanent part of your file, safe from being erased by ESC or by the left-arrow key. You can then type I to reenter Insert mode and type more text.

Now for our program. With the INSERT prompt line showing, press the RETURN key a couple of times, to move the cursor down, and then type

```
        PRORAFM DEMO;
```

You can use any name for your program, but in this discussion it will be called DEMO . Now press CTRL-C (type C while holding down the CTRL key). Your insertion so far is made "permanent", and the EDIT prompt line reappears. But, horrors! You made several typing errors when typing the word PROGRAM . Since you have already pressed CTRL-C , it is too late to backspace over your errors and retype them.

Fortunately, there are other ways. First, let's correct the missing G in PROGRAM . Using the left-arrow key, move the cursor left until it is sitting directly on the R . Then type I to reenter Insert mode. Ignore the fact that the remainder of the line seems to have suddenly disappeared, and type the missing letter G . When you press CTRL-C to make this insertion permanent, the rest of the line returns:

```
        PROGRAFM DEMO;
```

The letter F is certainly not needed, so move the cursor right (using the right-arrow key) until it is sitting directly on the F . Now type D to select the Editor's D(LETE option. When the DELETE prompt line appears,

```
>DELETE: < > <MOVING COMMANDS> [<ETX> TO DELETE, <ESC> TO ABORT]
```

press the right-arrow key once. The offending F instantly disappears. In Delete mode, moving the cursor in any direction deletes text. If you move the cursor back again, the deleted text reappears. What happens next is similar to Insert mode: if you press the ESC key, the deletion is forgotten, as if it had never happened. If you press CTRL-C, the deletion is made a permanent part of your file. To remove that F permanently, press CTRL-C. The line closes in to fill the deleted letter's place:

```
        PROGRAM DEMO;
```

Now you know how to use the Editor's Insert and Delete modes to write text and to correct your errors. Try typing the rest of program DEMO into your file. Be sure to "accept" your insertions, from time to time, by pressing CTRL-C . That way, you minimize your loss if you accidentally press the ESC key. Here is the complete program:

```
      PROGRAM DEMO;

        USES TURTLEGRAPHICS, APPLESTUFF;
        VAR ANGLE, DISTANCE : INTEGER;

        PROCEDURE CRAWL;
        BEGIN
          MOVE (2 * DISTANCE);
          TURN (ANGLE)
        END;

        BEGIN
          ANGLE := Ø;
          REPEAT
            INITTURTLE;
            PENCOLOR (WHITE);
            FOR DISTANCE := 1 TO 99 DO CRAWL;
            ANGLE := ANGLE + 5
          UNTIL KEYPRESS;
          TEXTMODE
        END.
```

When you are typing this program, the punctuation and spelling must be exactly as shown. The indentation of the lines is not important, but it easier to read as shown. You will notice that, once you have started a new indentation, the Editor maintains that indentation for you. To move back to the left, just press the left-arrow key before you type anything on the new line.

Program DEMO makes use of graphics routines in the Unit TURTLEGRAPHICS, and uses the keypress function from the Unit APPLESTUFF (see Chapter 7 for details). The third line of the program declares two integer variables, DISTANCE and ANGLE. Next, a Pascal procedure named CRAWL is defined, between the first BEGIN and END; . From here on, each time this new Pascal statement CRAWL is used, a graphics "turtle" will trace a line on the screen, of length 2*DISTANCE moving in the current direction, and will then change the direction by an amount ANGLE.

The next BEGIN and the last END. outline the main program. The portion of the program from REPEAT to UNTIL KEYPRESS is repeated over and over again, until any key on the Apple's keyboard is pressed.

In each repetition, the screen is cleared and the tracing color is set to WHITE. Then the procedure CRAWL is performed, first with the value of DISTANCE set to one, then with DISTANCE set to the value two, and so on, until DISTANCE is set to 99 . The "turtle" moves, then turns, then moves some more, then turns again, and so on, for 99 steps. That completes one design on the screen. In the next repetition, if no key has been pressed, the ANGLE has increased by 5 degrees, the screen is cleared by INITTURTLE, and the whole process starts again.

Now you should save this program. With the EDIT prompt line showing, type Q to select the Q(UIT option. The following message appears:

```
>QUIT:
     U(PDATE THE WORKFILE AND LEAVE
     E(XIT WITHOUT UPDATING
     R(ETURN TO THE EDITOR WITHOUT UPDATING
     W(RITE TO A FILE NAME AND RETURN
```

Type U to create a "workfile" diskette copy of your program (future versions of this file will be "Updates)". This workfile is a file on your boot diskette (APPLE1:) called SYSTEM.WRK.TEXT . The computer says

```
WRITING..
YOUR FILE IS 33Ø BYTES LONG.
```

(the number of bytes may be a little different) and then the COMMAND prompt line reappears. Now type R to select the R(UN option. This automatically calls the Compiler for you, since the workfile contains text. The disk drive containing APPLE2: whirrs and, if you have typed the program perfectly, the following messages (again, perhaps with slightly different numbers) appear, one by one:

```
COMPILING...


PASCAL COMPILER II.1 [B2B]
<   Ø>....
TURTLEGR [ 2483 WORDS]
<   5>.........................
APPLESTU [ 1Ø78 WORDS]
<  3Ø>..................
CRAWL    [ 1Ø98 WORDS]
<  46>.....
DEMO     [ 11Ø9 WORDS]
<  51>........
59 LINES
SMALLEST AVAILABLE SPACE = 1Ø98 WORDS
```

If the Compiler discovers mistakes, it will give you a message such as

```
PROFRAM <<<<
LINE 2, ERROR 18: <SP>(CONTINUE), <ESC>(TERMINATE), E(DIT
```

Don't despair; just type E for E(DIT . Your workfile will be automatically read back into the Editor for repairs. Read the error message at the top of the screen, press the spacebar, and make any necessary changes using I(nsert and D(elete. Then Q(uit, U(pdate the workfile, and R(un your program again, by typing Q U R (the Apple will store up several commands in advance).

When your program has been successfully Compiled, it is automatically executed. You will see the message

```
RUNNING...
```

and then a horizontal line appears on the screen. That is the first design your program draws: the white "turtle" moves out a distance 2*1 , turns an angle Ø ; moves 2*2 , turns Ø ; moves 2*3 , turns Ø ; etc. Keep watching as successive designs turn through larger and larger angles between moves. When you want to interrupt the program, press any key on the keyboard. You can R(un the program again at any time, by typing R . Since the latest version of your program has already been compiled, it will be executed immediately, this time.

Try making changes to the program, by setting a different starting ANGLE, or a different increment to the ANGLE, or a different distance to MOVE. To do this, type E for E(DIT, use I(nsert and D(elete to make changes, and then Q(uit, U(pdate the workfile, and R(un again by typing Q U R . This cycle of Edit-Run-Edit-Run is the basis of all program development in the Apple Pascal system.

The workfile on APPLE1: now contains the text version of your program in a file named SYSTEM.WRK.TEXT , and the compiled P-code version of your program in another file named SYSTEM.WRK.CODE . When your program is running as you want it to, you should save the text and code workfile under other filenames. With the COMMAND prompt line showing, type F to enter the Filer. When the FILER prompt line appears, place in any available drive the diskette on which you want your program stored. Then type S for S(ave. You will be asked

```
SAVE AS ?
```

and you should respond by typing the name of the destination diskette, followed by a colon, followed by any filename with ten or fewer characters. For example, you might type

```
MYDISK:DEMO
```

When you press the RETURN key, the boot diskette's workfile, SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE, is saved on MYDISK: under the filenames DEMO.TEXT and DEMO.CODE . These messages will tell you what has happened:

```
APPLE1:SYSTEM.WRK.TEXT
 --> MYDISK:DEMO.TEXT
APPLE1:SYSTEM.WRK.CODE
 --> MYDISK:DEMO.CODE
```

WHAT TO LEAVE IN THE DRIVES

When you turn the Apple off, it is a good idea to leave the diskette called APPLE1: in disk drive #4: . If there is no diskette or some other diskette in #4: when the Apple is accidentally turned on, the drive will spin the disk indefinitely. If this continues for hours and hours, some wear will take place on the diskette and the drive. So, it is a good idea to make a habit of leaving a copy of APPLE1: (now that you have copies) in #4: when you turn the system off.

Of course, if you turn on the system and APPLE1: is not in #4:, just press the key marked RESET . Place APPLE1: in #4: and turn the system off and then on again. No damage results from turning on the computer with the wrong diskette (or no diskette) in the drive. Gradual, unnecessary wear results from leaving the disk drive running for a long period of time with the incorrect diskette (or no diskette) in the drive.

USING MORE THAN TWO DRIVES

The primary difference between using a two-drive system and using larger systems is that you rarely need to remove APPLE1: from its usual location in drive #4: , and can do all copying and transfering between files in the other drives.

For example, with four drives, you can have APPLE1: in #4:, APPLE2: in #5:, and APPLE3: in #11:; then you can format diskettes by placing them in #12:, without having to remove any of the system diskettes.

A one-drive system is a useful tool for learning Pascal and running programs written on other systems. A one-drive system can, in fact, do anything that the larger systems can do, up to the limits of the actual storage space available. For software development of any magnitude, however, two drives are recommended. Again, more drives make life easier. Word processing, using the text editor, is most pleasant with a three-drive system. Some business applications, which can benefit from having over half a megabyte on line, might use six drives.

No specific instructions will be given here on using multiple-drive systems. Acquaintance with a two-drive system should be sufficient introduction.

MULTIPLE-DRIVE SUMMARY

STARTING UP THE SYSTEM

To start the system, place diskette APPLE1: in disk drive #4: (slot 6, drive 1); then turn on the Apple's power. When the "WELCOME" message appears, Pascal is running.

FORMATTING NEW DISKETTES

To format a new diskette, have Pascal's COMMAND prompt line showing. Place diskette APPLE3: in any drive except #4: , and type

```
        X
```

Now, in response to the query

```
        EXECUTE WHAT FILE?
```

type

```
        APPLE3:FORMATTER
```

When the question:

```
        FORMAT WHICH DISK ?
```

appears, place the new diskette in any drive except #4: , and then type the number of that drive. For example, if you put the new diskette in drive #5: , type

```
        5
```

When you press the RETURN key, the diskette will be formatted. To leave the formatting program, press the RETURN key in response to the question WHICH DISK ? A newly formatted diskette has the name BLANK:

COPYING DISKETTES

To copy a diskette, have the COMMAND prompt line showing, and put APPLE1: in drive #4: . Get into the Filer by typing

```
        F
```

Once the FILER prompt line is showing, you may remove APPLE1: from its drive if you need to. Put the source diskette you wish to copy into one drive, and the destination diskette you want to copy onto into another drive, then type

```
        T
```

Now, when this question appears:

```
        TRANSFER ?
```

reply by typing the name of the source diskette to be copied, and then press the RETURN key. For example, you might type

```
        APPLE3:
```

Now, when the next question appears:

```
        TO WHERE ?
```

reply with the name of the destination diskette that is to become the backup copy. For example, you might type

```
        BLANK:
```

Lastly, you will be asked

```
        TRANSFER 28Ø BLOCKS ?
```

and

```
        DESTROY BLANK: ?
```

Reply

```
        Y
```

to both, and BLANK: will be turned into a perfect copy of APPLE3: . Be sure to put diskette APPLE1: back into drive #4: before Q(uitting the Filer.

EXECUTING A PROGRAM

To execute a previously compiled program, put APPLE1: in drive #4: and put the diskette containing the program file into any other drive. With the COMMAND prompt line showing, type X for X(ecute. When the computer prompts

```
        EXECUTE WHAT FILE?
```

answer by typing the name of the diskette and codefile you wish to have executed. DO NOT type the .CODE suffix. For example, to execute the program GRAFDEMO.CODE on diskette APPLE3: , you would type

```
        APPLE3:GRAFDEMO
```

The program should now run.

WRITING A PROGRAM

To start a new file in the Editor, put APPLE1: in drive #4: and put APPLE2: in drive #5: . With the COMMAND prompt line showing, type F to enter the Filer. Then type N for N(ew. If you are asked

```
        THROW AWAY CURRENT WORKFILE ?
```

type Y for Y(es. When you see the message

```
        WORKFILE CLEARED
```

type Q to Q(uit the Filer, and then type E to enter the Editor. This message appears:

```
   >EDIT:
   NO WORKFILE IS PRESENT. FILE? ( <RET> FOR NO FILE <ESC-RET> TO EXIT )
```

Press the RETURN key, and the full EDIT: prompt line appears. You can now insert text at the cursor position by typing I for I(nsert and then typing your program. Conclude each insertion by pressing CTRL-C. Delete text at the cursor position by typing D for D(elete and then moving the cursor to erase text. Conclude each deletion by pressing CTRL-C . When you have written a version of your program, type Q to Q(uit the Editor, and then type U to U(pdate the workfile to contain your latest program version.

With the COMMAND prompt line showing, you can then type R to R(un your program. This automatically compiles the text workfile (using the Compiler program on APPLE2:), stores the compiled code workfile, and executes it. To reenter the Editor, type E in response to the COMMAND prompt. The text workfile is automatically read back into the computer.

When a version of your program is complete, you can U(pdate the text workfile to contain that latest version and R(un the program to create a code workfile of that version. To save the workfile versions of your program on another diskette for later use, place that diskette in drive #5: and type F in response to the COMMAND prompt to enter the Filer. Then type S for S(ave. When you see the prompt

```
        SAVE AS ?
```

type the name of the diskette and file where you want your program saved. Do not type any .TEXT or .CODE suffix. For example, if you want your program saved under the filename DEMO on the diskette MYDISK: , you might type

```
        MYDISK:DEMO
```

The text workfile SYSTEM.WRK.TEXT on APPLE1: is saved as DEMO.TEXT on MYDISK:, and the code workfile SYSTEM.WRK.CODE is saved as DEMO.CODE on MYDISK: .

APPENDIX F
APPLE PASCAL SYNTAX

These diagrams represent all of the syntax of Apple Pascal. However, they do not show the semantic rules. To understand the distinction between syntax and semantics, consider the sentence "John Smith is a citizen of the three of clubs." This sentence is correct syntactically (i.e., grammatically) but wrong semantically -- the three of clubs is not something one can be a citizen of.

Similarly, the diagram for a statement shows that one kind of statement is an identifier optionally followed by one or more expressions in parentheses. The diagram does not show the semantic restriction, which is that the identifier must be the identifier of a procedure. Some of the important semantic restrictions are given in the notes accompanying the diagrams.

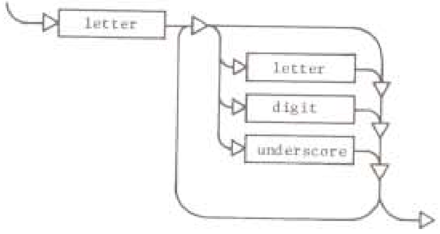
With this limitation in mind, you will find that the diagrams are useful as reference material. To read one of these diagrams, start at the left and follow arrows until you come out at the right. Whenever the arrows branch, you can go either way. Any path that goes through from the left to the right defines a syntactically correct Apple Pascal construction.

Circles and ovals are used to enclose characters and words that are to be typed exactly as shown; for example, the word NIL in the diagram for an unsigned constant. Boxes with square corners enclose words and phrases that stand for something else; for example, the word "letter" in the diagram for an identifier stands for any letter.

The vertical arrow symbol used in these diagrams corresponds to the "~" character in the text of this document and on the Apple keyboard.

A word or phrase that you find in a square-cornered box is the title of another diagram; the diagram shows what the word or phrase can stand for when it appears in other diagrams. (Exceptions: there are no diagrams for "letter," "digit," and "underscore.")

identifier

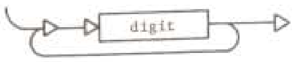


1. The letters are a..z and A..Z .

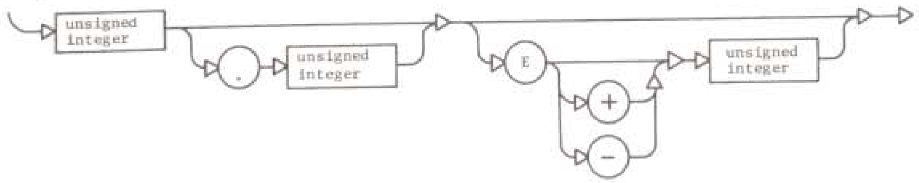
2. The digits are Ø..9 .

3. The underscore character, _ , is not available on the Apple keyboard. However some external terminals provide it.

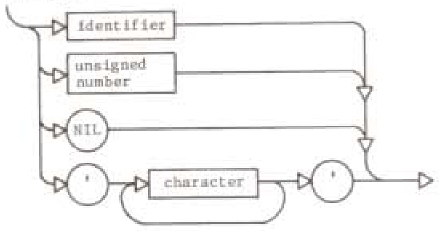
unsigned integer



unsigned number



unsigned constant



1. The identifier in this diagram must be the identifier of a constant.

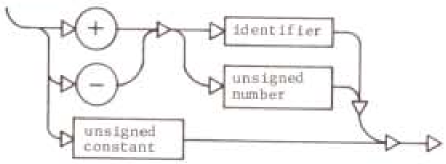
2. The bottom line of the diagram represents a string constant. A single apostrophe cannot appear as a character in the string constant, since this would end the constant. However, you can place two consecutive apostrophes in the string constant, and the result will be a single apostrophe in the value of the string. For example:

```
     WRITELN('DON''T FORGET TO BOOGIE!')
```

will cause the following output:

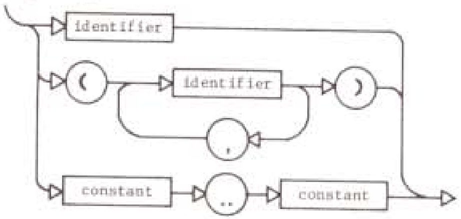
```
     DON'T FORGET TO BOOGIE!
```

constant



The identifier in this diagram must be the identifier of a constant.

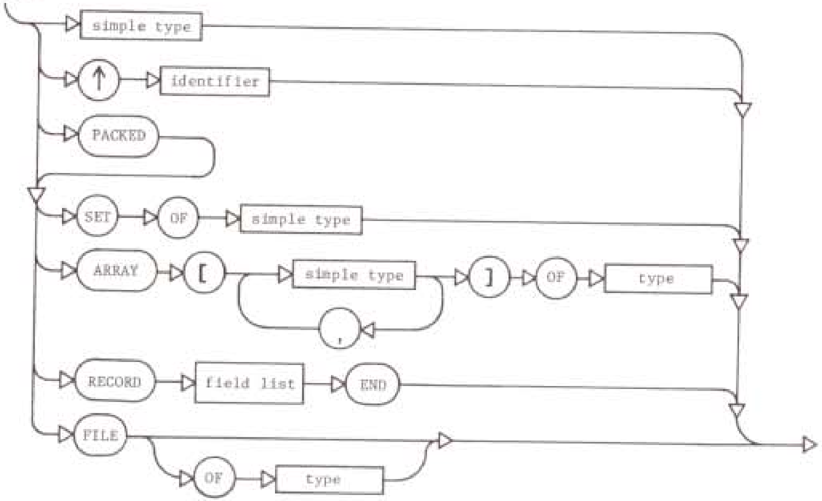
simple type



1. The identifier in the top line of this diagram must be the identifier of a type.

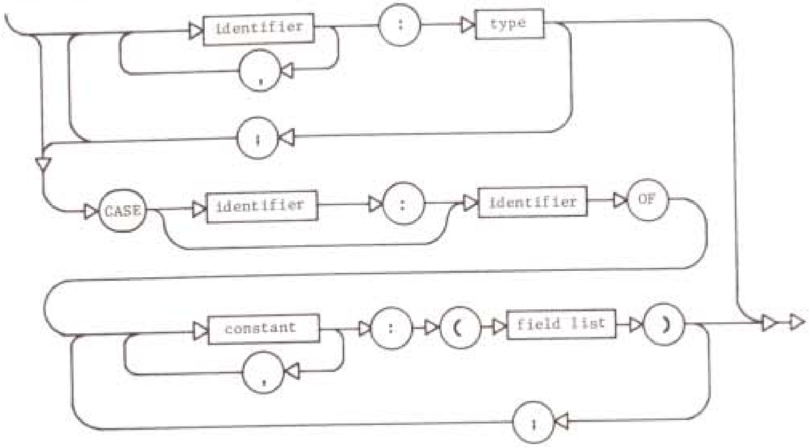
2. The identifier(s) in the second line define a scalar type. They are being declared, so they must be identifiers that are not yet declared or predefined.

type



The identifier in this diagram must be the identifier of a type.

field list

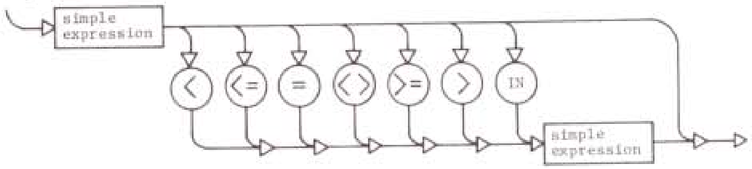


1. The identifier(s) in the top line are being declared, so they must be identifiers that are not yet declared or predefined.

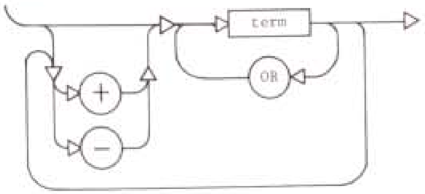
2. The identifier between the word CASE and the colon is the tag field. It is being declared, so it must be an identifier that is not yet declared or predefined.

3. The identifier between the colon and the word OF must be the identifier of a type.

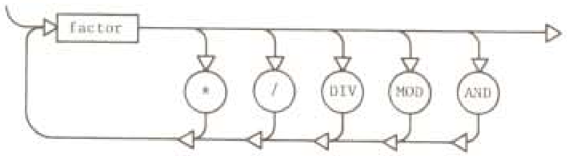
expression



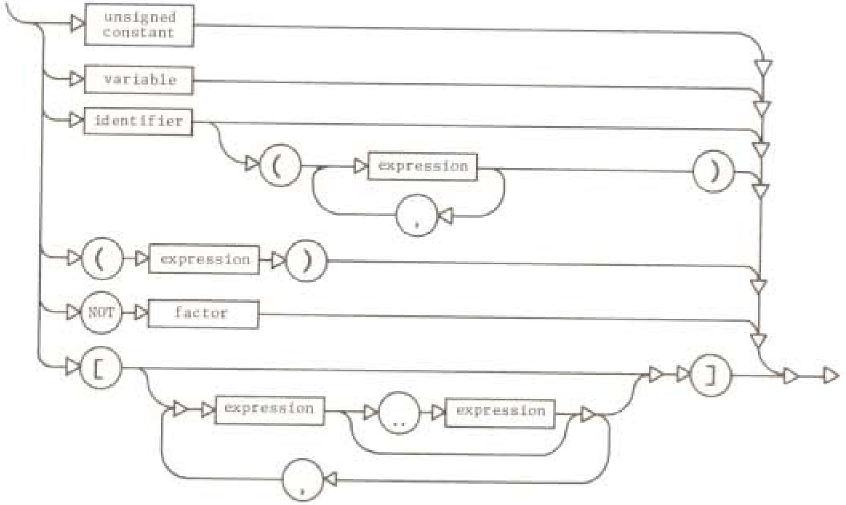
simple expression



term



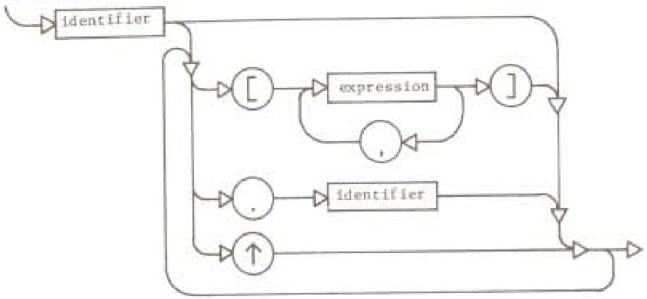
factor



1. The identifier in this diagram must be the identifier of a function.

2. The bottom portion of the diagram (square brackets and expressions) indicates the formation of a set. The values of the expressions must be of the same underlying type.

variable

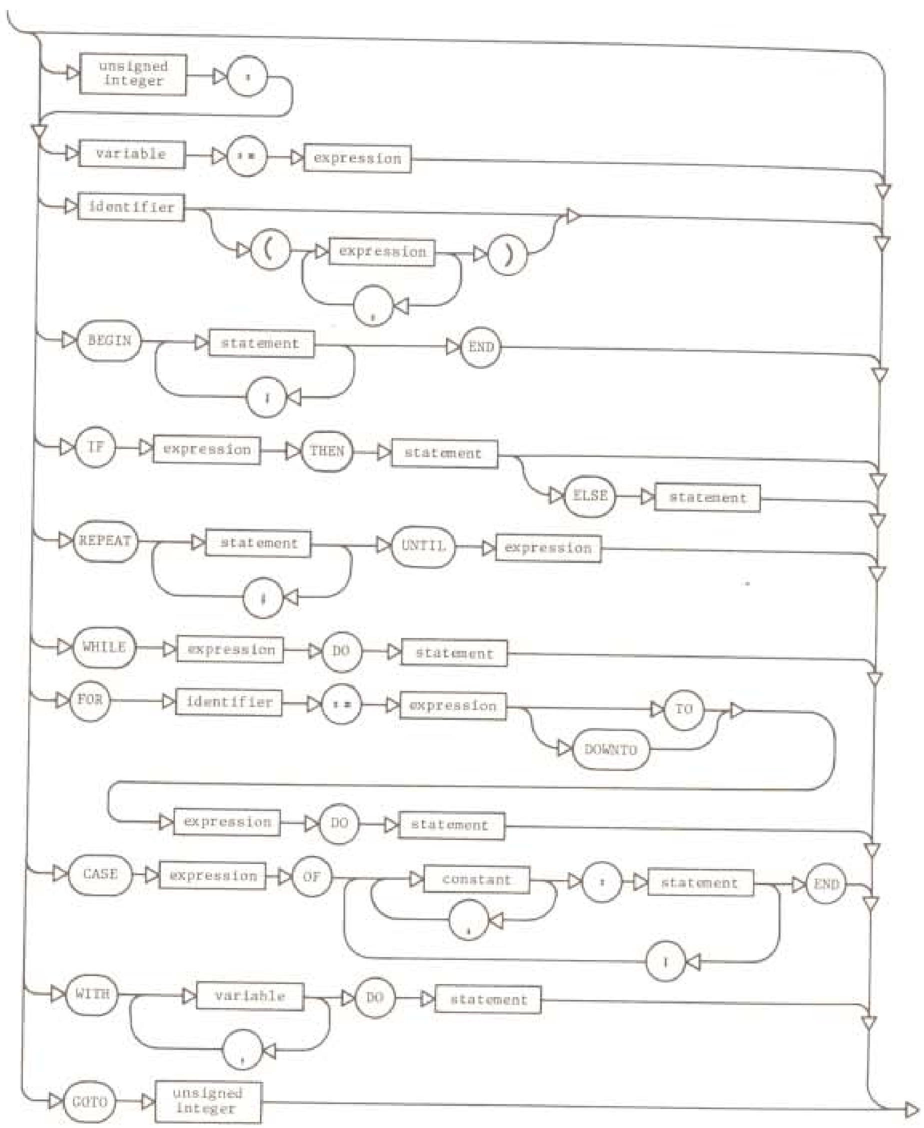


1. If the identifier at the top of the diagram is that of an array, the expression(s) in square brackets may be used to subscript it. The values of the expressions(s) must be compatible with subscript types declared for the array.

2. If the identifier at the top of the diagram is that of a record, it may be followed by a period and a second identifier. The second identifier must be the identifier of one of the fields of the record.

3. If the identifier at the top of the diagram is that of a pointer, it may be followed by the up-arrow character.

statement



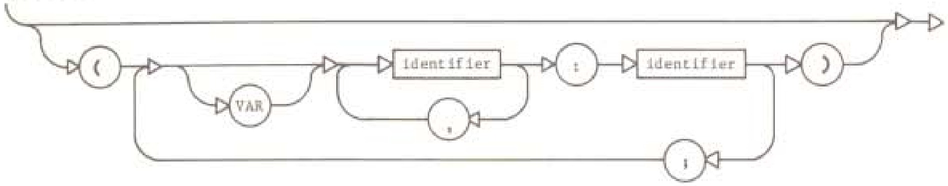
1. Note that there is a "null" path through this diagram, across the top and down the right-hand side without including anything. This represents what happens when a superfluous semicolon occurs in a program.

2. The unsigned integer at the top of the diagram is a label, and must have been declared in a LABEL declaration.

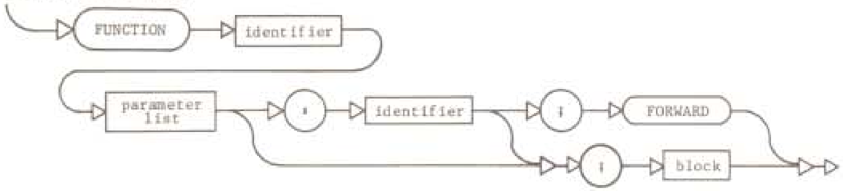
3. The identifier in the third line of the diagram (above BEGIN) must be the identifier of a procedure.

4. The expression in an IF, REPEAT, or WHILE statement must have a BOOLEAN value.

parameter list



function declaration



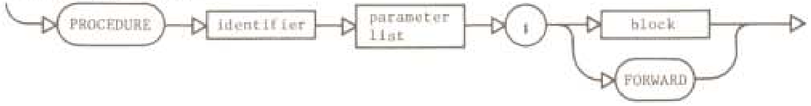
This diagram shows all of the forms a function declaration can take:

- The normal form includes a parameter list (which may be null) and the colon followed by an identifier (which must be that of a type). The declaration ends with a block.

- The FORWARD declaration is like the normal form except that the word FORWARD is used instead of a block.

- Following a FORWARD declaration, the function declaration has no parameter list or type identifier and ends with a block.

procedure declaration



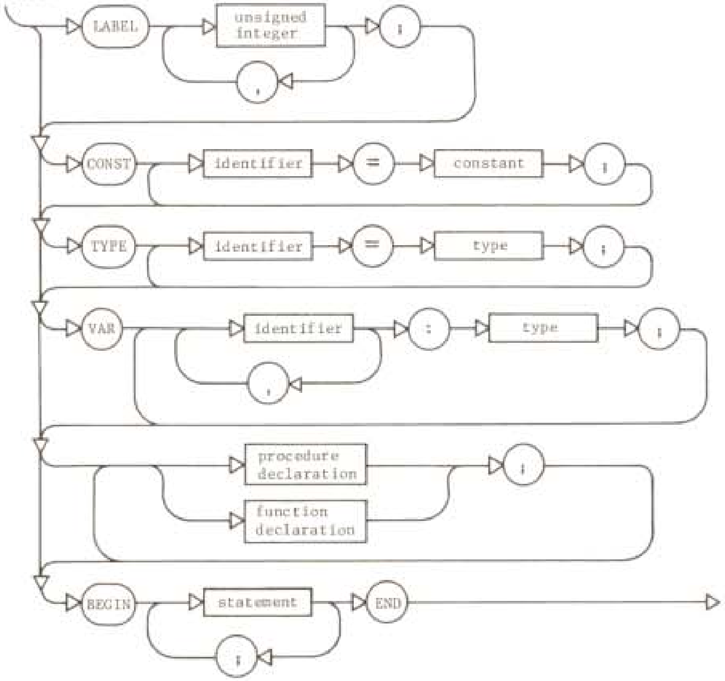
This diagram shows all of the forms a procedure declaration can take:

- The normal form includes a parameter list (which may be null). The declaration ends with a block.

- The FORWARD declaration is like the normal form except that the word FORWARD is used instead of a block.

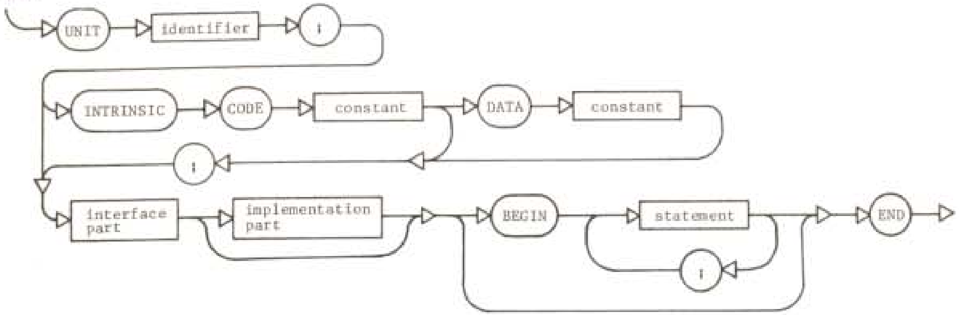
- Following a FORWARD declaration, the procedure declaration has no parameter list and ends with a block.

block



This is one of the fundamental structural units: it contains all the local data declarations (except parameters) and all the statements for one program, procedure, or function.

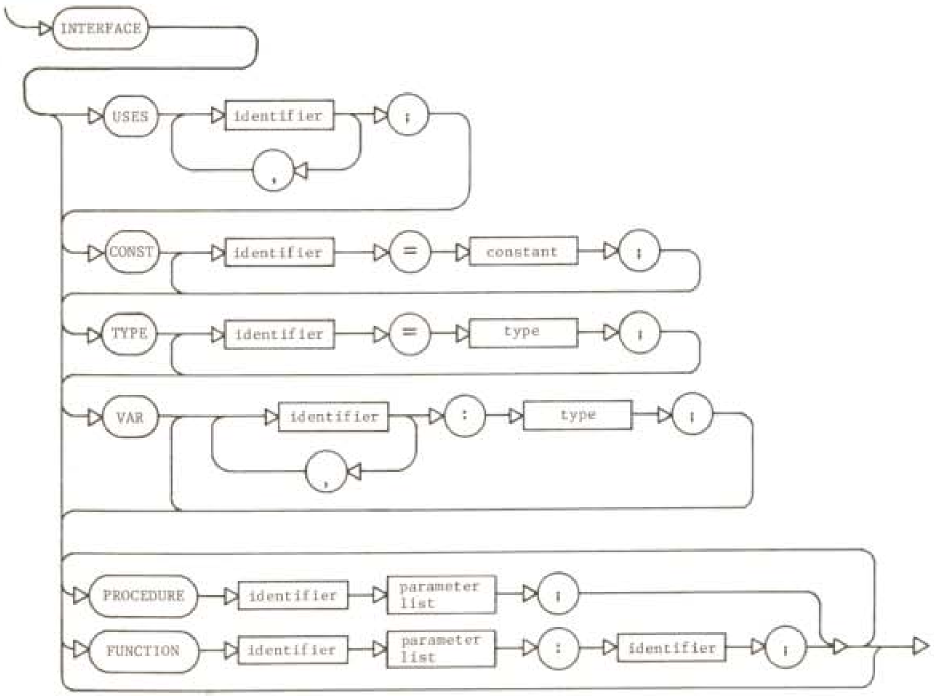
unit



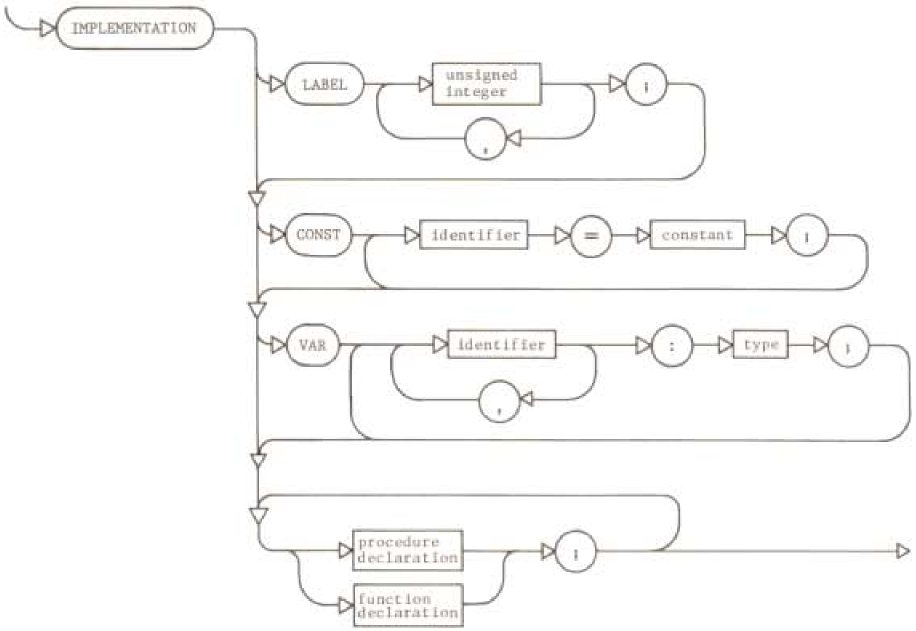
1. In an intrinsic unit, the constants following CODE and DATA must be integers and should be carefully chosen.

2. The words BEGIN and END with the statements between them are the "initialization" of the unit.

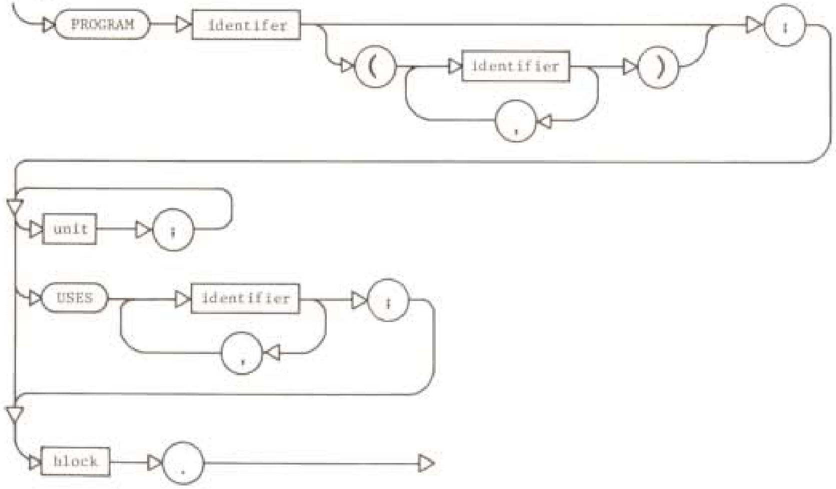
interface part



implementation part



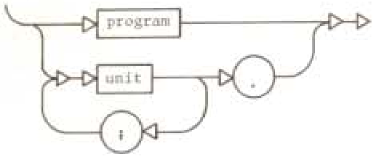
program



1. The program heading may contain identifiers in parentheses in accordance with Standard Pascal syntax. However the identifiers are ignored.

2. Note that any units defined in the program must immediately follow the program heading. This would normally be done only for test purposes.

compilation



A compilation is simply something that the compiler can compile. This may be a program (which may contain units), or one or more units separated with semicolons and ending with a period.

INDEX