PL/I-80 APPLICATIONS GUIDE SUPPLEMENT

This is a supplement to the PL/I-80 Applications Guide. The primary purpose of this section is to provide additional information for the proper application of PL/I-80 to commercial processing problems, with an emphasis on decimal computations and the use of picture formatted output. Revision level 1.3 (or later) of the compiler and runtime system is required for compiling and executing the programs listed in this supplement.

12. COMMERCIAL PROCESSING USING PL/I-80.

The purpose of this section is to familiarize you with some techniques used in PL/I-80 in processing commercial data. In particular, the various decimal arithmetic operations are described in some detail. Conversion between Fixed Decimal and Floating Point Binary is examined, including the use of the ftc (float to character) library function. The discussion also includes examples of picture formatted output, along with a presentation of precision and scale evaluation when using the four basic arithmetic functions with decimal operands. Four programs are presented which typify the use of decimal operations in actual applications.

12.1. A Comparison of Decimal and Binary Operations.

We have been taught from childhood to perform arithmetic operations using base ten arithmetic where the permissible digits range from 0 through 9. Further, application languages such as Basic, Fortran, Cobol, and PL/I allow us to write programs which process base ten constants and data items in simple and readable forms. Internally, however, computers generally perform the arithmetic operations using either binary or decimal numbers. Binary numbers are more "natural" for internal computer arithmetic since the 1's and 0's can be directly processed by the on-off electronic switches found in arithmetic processors. Because our programs generally process decimal values, it becomes necessary to convert into a binary form on input and back to a decimal form on output. As we shall see below, this conversion can introduce truncation errors which are unacceptable in commercial processing. Thus, decimal arithmetic is often required in order to avoid the propagation of errors throughout computations.

In most languages, the programmer has no control over the internal format used for numeric processing. In fact, two of the most popular Basic interpreters for microprocessors differ primarily in the internal number formats. One uses floating point binary, while the other performs calculations using decimal arithmetic. Pascal language translators generally use floating and fixed point binary formats with implementation-defined precision, while Fortran always performs arithmetic using floating or fixed point binary. Cobol, on the other hand, was designed for use in commercial applications where exact dollars and cents must be maintained throughout computations, and thus data items are processed using decimal arithmetic.

PL/I-80 gives the programmer the choice between representations so that each program can be tailored to the exact needs of the particular application. Fixed Decimal data items are used in PL/I-80 to perform commercial functions, while Float Binary items are used for scientific processing where computation speed is the most important factor. The two programs shown below illustrate the essential difference between the two computational forms:

```
dec_comp:
    proc options(main);
    dcl
        i fixed,
        t decimal(7,2);
    t = 0;
        do i = 1 to 10000;
        t = t + 3.10;
        end;
    put edit(t) (f(10,2));
    end decimal_comp;
```

```
bin_comp:
    proc options(main);
    dcl
        i fixed,
        t float(24);
    t = 0;
        do i = 1 to 10000;
        t = t + 3.10;
        end;
    put edit(t) (f(10,2));
    end bin_comp;
```

The two programs perform the simple function of summing the value 3.10 a total of 10,000 times. The only difference between these programs is that "dec_comp" computes the result using a Fixed Decimal variable, while "bin_comp" performs the computation using Float Binary. Dec_comp produces the correct result 31000.00, while bin_comp produces the approximation 30997.30. The difference is due to the inherent truncation which occurs when certain decimal constants, such as 3.10, are converted to their binary approximations. Since no conversion occurs when Fixed Decimal variables are used, dec_comp produces an exact result.

These two programs can be considered simplifications of a more general situation: suppose Chase-Manhattan Bank has processed 10,000 deposits of $3.10 during a particular day. Using a program based upon Floating Binary, there would be an extra $2.70 unaccounted for at the end of the day (there have been cases where crooked systems programmers have been caught redirecting the "excess cash" produced by such errors into their own accounts!). This is due to the fact that .10 cannot be represented as a finite binary fractional expansion. That is, 3.10 is actually approximated as 3.099999E+00 in Float Binary form. Each addition propagates a small error into the sum, resulting in an incorrect total. In scientific applications, the inherent truncation errors are often insignificant and thus ignored. In commercial applications such inherent errors are unacceptable.

It should be noted that there are situations where decimal arithmetic also produces truncation errors which can propagate throughout computations. The expression 1/3, for example, cannot be represented as a finite decimal fraction, and thus is approximated as

0.3333333 ...

to the maximum possible precision. However, due to our life-long experience with decimal computations, we expect such errors to occur and adjust our programming to account for the situation. In fact, we know that such errors will only occur when explicit division operations take place. We expect that 1/10 will be represented exactly as .10, and not just a close approximation. But herein lies the difficulty with Float Binary representations: some decimal constants which can be expressed as finite fractional expansions in Fixed Decimal cannot be written as finite binary fractions and thus are necessarily truncated during conversion to Float Binary form.

With this introduction, we will now proceed to explain exactly how Fixed Decimal numbers are represented and manipulated.

12.2. Decimal Computations in PL/I-80.

Fixed Decimal arithmetic can be performed in PL/I-80 programs. There are both advantages and disadvantages in selecting Fixed Decimal arithmetic when contrasted to Floating Point formats. First, Fixed Decimal arithmetic guarantees that there will be no loss of significant digits. That is, all digits are considered significant in a computation so that multiplication, for example, will not truncate digits in the least-significant positions. Further, Fixed Decimal arithmetic precludes the necessity for exponent manipulation, and thus the operations are relatively fast when compared to alternative decimal arithmetic formats. The disadvantage, however, is that since all digits are considered significant, the programmer must keep track of the range of values that arithmetic operands can take on. The paragraphs which follow provide the necessary background to properly program using Fixed Decimal formats.

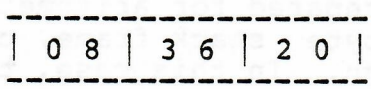
Decimal variables and constants in PL/I-80 have both "precision" and "scale." Precision denotes the number of digits in the variable or constant, while scale defines the number of digits in the fractional part. For Fixed Decimal variables and constants, the precision must not exceed 15 and the scale must not exceed the precision. The precision and scale of a PL/I-80 variable is defined in the variable's declaration:

```
declare x fixed decimal(10,3);
```

while the precision and scale of a constant are derived by the compiler by counting the number of digits in the constant, and the number of digits following the decimal point. The constant

```
-324.76
```

for example, has precision 5 and scale 2. Internally, Fixed Decimal variables and constants are stored as Binary Coded Decimal (BCD) pairs, where each BCD digit occupies either the high or low order 4-bits of each byte. The most significant BCD digit defines the sign of the number or constant, where 0 denotes a positive number, and 9 defines a negative number in 10's complement form, as described below. Since numbers are always stored into 8-bit byte locations, there may be an extra "pad" digit at the end of the number to align to an even byte boundary. The number 83.62, for example, is stored as



where each digit represents a 4-bit "half byte" position in the 8-bit

value. The leading BCD pair is stored lowest in memory.

Negative numbers are stored in 10's complement form to simplify arithmetic operations. A 10's complement number is similar to a 2's complement binary representation, except the complement value of the digit x is 9-x. To derive the 10's complement value of a number, form the complement of each digit (by subtracting the digit from 9), and add 1 to the final result. Thus, the 10's complement of -2 is formed as follows:

```
9 - 2 + 1 = 7 + 1 = 8
```

The sign digit is attached to this number, and internally carried as the single-byte value

```
-------
| 9 8 |
-------
```

Note, for example, that you can add -2 and +3 as follows

```
98 + 03 = 101
```

The carry-out beyond the sign digit is ignored, and the correct result 01 is produced through the addition. For this reason, addition and subtraction in PL/I-80 are equivalent: in the case of subtraction, the subtrahend is first complemented and the addition operation is applied. In all cases, numeric values are sign-extended to 15 digits before arithmetic operations are applied. For convenience of notation, negative numbers will be shown with a leading "-" sign, with the assumption that the underlying representation is 10's complement form. Thus, the number shown above will be written as

```
-------
| - 2 |
-------
```

It should be noted that there is no need to explicitly store the decimal position in memory, since the precision and scale for each variable and constant is known by the compiler. Before each arithmetic operation, the compiled code causes the necessary alignment of the operands. In later examples, however, a decimal point position is often shown in order to more easily determine the effect of alignment. The number -324.76 may be shown, for example, as

```
-------------------
| - 3 | 2 4 | 7 6 |
-------------------
        ^
```

When this value is prepared for arithmetic processing, it is first loaded into an 8-byte stack frame, consisting of 15 decimal digits with a high-order sign. In this case, the -324.76 is shown as

A convenient model for discussing the various arithmetic operations is to visualize a 15-digit mechanical or electronic calculator with a hand-movable decimal point. At the beginning of each operation, you must properly line-up the operands for the arithmetic operation and, upon completion of the operation, you must decide where the resulting decimal point appears. Actually, the compiler performs the alignment and accounts for the decimal point position, but it's useful for you to imagine what is taking place so that you can avoid overflow or underflow conditions. In some cases, you may wish to force a precision and/or scale change during the computation using the DECIMAL or DIVIDE built-in functions. Examples of such functions are given in the sample programs discussed in the sections which follow.

First, we'll examine each of the arithmetic functions in order to determine the alignment, precision, and scale which occurs in each case.

12.3. Addition and Subtraction.

As mentioned above, addition and subtraction are functionally equivalent in PL/I-80, since subtraction is accomplished by forming the 10's complement of the subtrahend and then performing an addition. Given two numbers x and y with precision and scale (p,q) and (r,s), respectively, the addition operation proceeds as follows. First, the two operands are loaded to the stack and aligned. Alignment takes place by shifting the operand with the smaller scale to the left until the decimal positions are the same. Given that the scale of x is greater than the scale of y, y is shifted q-s positions to the left, with zero values introduced in the least signficant positions. After alignment, y has precision r+(q-s) and scale q. (A Fixed Overflow condition is signalled if significant digits are shifted into the sign position during the alignment process.)

In order to provide a specific example, suppose x = 31465.2437 and y = 9343.412 so that x has precision p = 9 and scale q = 4, while y has precision r = 7 and scale s = 3. Before alignment, the numbers appear as

```
                        |<----- p=9 ----->|
                                  |< q=4 >|
x = + 0 0 0 0 0 0 3 1 4 6 5 2 4 3 7
                                  ^

y = + 0 0 0 0 0 0 0 0 9 3 4 3 4 1 2
                          |       ^       |
                          |       |<s=3>|
                          |<--- r=7 --->|
```

The value y is aligned with x by shifting q-s = 4-3 = 1 positions to the left, producing

```
                        |<----- p=9 ----->|
                                  |< q=4 >|
x = + 0 0 0 0 0 0 3 1 4 6 5 2 4 3 7
                                  ^

y = + 0 0 0 0 0 0 0 0 9 3 4 3 4 1 2 0
                          |       ^       |
                          |       |<  q  >|
                          |< r+(q-s) = 8 >|
```

Note that the number of digits in the whole part of x is p-q, while the whole part of y contains r-s digits:

```
|< p-q=5 >|
 3 1 4 6 5
   9 3 4 3
  |<r-s=4>|
```

so the sum must contain p-q=5 digits in the whole part:

```
  3 1 4 6 5
+   9 3 4 3
-----------
  4 0 8 0 8
 |< p-q=5 >|
```

Note, however, that sufficiently large values could produce an overflow, requiring one extra digit in the whole part:

```
  9 9 9 9 9
+ 9 9 9 9 9
-----------
1 9 9 9 9 8
|<(p-q)+1=6>|
```

Thus, the total number of digits in the sum of x and y is the number of digits in the whole part, (p-q)+1=6, plus the number of digits in the fraction, given by q, resulting in a precision of

$$(p-q)+1 + q = p + 1$$

Given two values x and y of arbitrary precision and scale, we can use the specific case shown above to derive the general form of the resulting precision and scale. First, the scale must be the

greater of q and s, given by

$$\max(q,s)$$

and thus, the resulting precision must have max(q,s) fractional digits. Second, the whole part x contains p-q digits, while the whole part of y contains r-s digits. The result contains the larger of p-q and r-s digits, plus the fractional digits, along with one overflow digit, or a total of

$$\max(p-q,r-s) + \max(q,s) + 1$$

digit positions. Since the precision cannot exceed 15 digits in PL/I-80, the resulting precision must be

$$\min(15,\max(p-q,r-s)+\max(q,s)+1)$$

digits. Written as a pair, the precision and scale of the resulting addition or subtraction is

$$(\min(15,\max(p-q,r-s)+\max(q,s)+1), \max(q,s))$$

Using the above example,

```
                          |<----- p=9 ----->|
                          |         |< q=4 >|
x = + 0 0 0 0 0 0 3 1 4 6 5 2 4 3 7
                            ^
y = + 0 0 0 0 0 0 0 9 3 4 3 4 1 2 0
                          |   ^     |
                          |   |< q >|
                          |< r+(q-s) = 8 >|
      ---------------------------------
x + y = + 0 0 0 0 0 0 4 0 8 0 8 6 5 5 7
                        |     ^       |
                        |     |<- 4 ->|
                        |<------ 10 ------->|
```

the precision (10,4) shown in the diagram is derived using the expression

$$(\min(15,\max(9-4,7-3)+\max(4,3)+1), \max(4,3))$$

or

$$(\min(15,\max(5,4)+4+1), 4) = (\min(15,10),4) = (10,4)$$

12.4. Multiplication.

Evaluation of precision and scale for the result of multiplication is somewhat simpler than addition and subtraction since

no decimal point alignment is required before the multiplication is applied. The two operands x and y with precision and scale (p,q) and (r,s), respectively, are multiplied digit-by-digit to produce the result. Similar to simple hand-calculations, the number of decimal places in the result is the sum of the scale factors q and s. The number of digits in the result is the sum of the precisions of the two operands. To conform to the PL/I standard, however, one additional digit position is included in the final precision. Thus, the precision and scale of the result of multiplication is given by

(min(15,p+r+1),q+s)

Suppose, for example, that x = 924.5 and y = 862.33, yielding the precision and scale values (4,1) and (5,2):

```
x = + 0 0 0 0 0 0 0 0 0 0 0 9 2 4 5
                            ^

y = + 0 0 0 0 0 0 0 0 0 0 8 6 2 3 3
                          ^
```

The product of the digits of x and y are shown below with the resulting precision and scale:

```
x * y = + 0 0 0 0 0 0 7 9 7 2 2 4 0 8 5
                                ^
                                |< 3 >|
                        |<------- 10 ------>|
```

where the precision is computed as

(min(15,4+5+1),1+2) = (min(15,10),3) = (10,3)

The Fixed Overflow condition is signalled if the product contains more than fifteen significant digits. In the example of the previous section, x = 31465.2437 and y = 9343.412. The product x*y has precision (17,7) with 16 significant digits, resulting in Fixed Overflow. In this particular case, the DECIMAL function must be applied to reduce the number of significant digits in either x or y. The computation could be carried out as

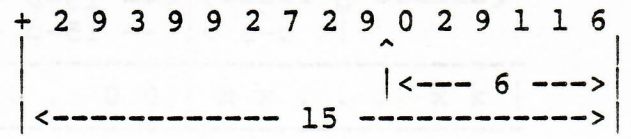
DECIMAL(x,9,3) * y

which loads the stack with the two values shown below before the multiplication takes place:

```
DECIMAL(x,9,3) = + 0 0 0 0 0 0 0 3 1 4 6 5 2 4 3
                                  ^

y = + 0 0 0 0 0 0 0 0 9 3 4 3 4 1 2
                      ^
```

The precision and scale of the product is



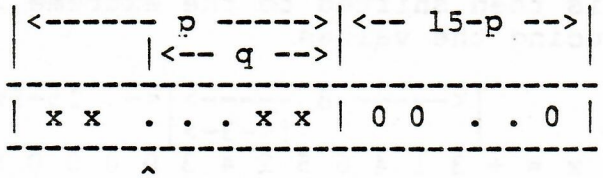
Note that the precision computation p+r+1 produces the value 16 which is then reduced to PL/I-80's maximum 15 digit precision by

```
min(15,p+r+1) = min(15,16) = 15
```

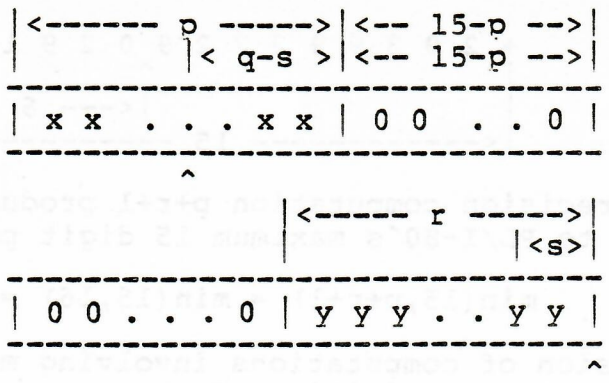
Since the precision of computations involving multiplications can grow rapidly, it is the responsibility of the programmer to ensure that the precisions of the operands involved will not produce overflow. Again, precision can be explicitly declared with the variables involved in the computation, or the DECIMAL function can be applied to reduce the precision of a temporary result.

12.5. Division.

The division operation is the only one of the four basic arithmetic operations which may produce truncation errors, as described in Section 12.1. Thus, each division operation produces a maximum precision value, consisting of 15 decimal digits, with a resulting scale which depends upon the scale values of the two operands. Assume that x and y have precision (p,q) and (r,s), and that x is to be divided by y. The division operation proceeds as follows. First, x is shifted to the extreme left by introducing 15-p zero values on the right, leaving the dividend in the stack as



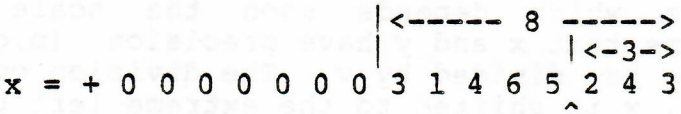
The decimal point of x is then effectively shifted right by an amount s to properly align the decimal point in the result, producing the operands



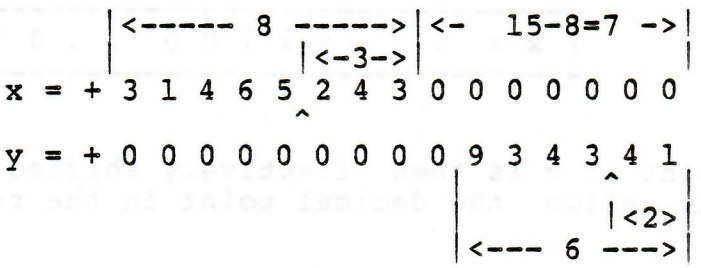
The significant digits of x are then continuously divided by the significant digits of y until 15 decimal digits are generated. Referring to the above diagram, note that the number of fractional digits produced by the division is determined by the placement of the adjusted decimal point in x. The field following the decimal point contains (q-s) plus (15-p) positions, yielding the following precision and scale for the result of the division

(15, (q-s)+(15-p)) or (15,15-p+q-s)

Suppose x = 31465.243, and y = 9343.41, yielding precision and scale values of (8,3) and (6,2), respectively. The value x when loaded appears as



The value of x is then shifted to the extreme left and the value of y is loaded, producing the values



The imaginary decimal points are shifted to the right by two positions in order to properly align the decimal point in the result, producing

```
                |<----- 8 ----->|<---- 7 ---->|
                |               |1|           |
x = + 3 1 4 6 5 2 4 3 0 0 0 0 0 0 0
                      ^
y = + 0 0 0 0 0 0 0 0 0 9 3 4 3 4 1
                                  |           ^
                                  |<--- 6 --->|
```

The significant digits of x are divided by the six significant digits of y, and the result is

```
                |<---------- 15 ------------>|
                                |<-- 1+7=8 -->|
x/y = + 0 0 0 0 0 0 0 3 3 6 7 6 4 0 1
                      ^
```

In this case, the precision and scale of the result is given by

(15,(15-p+q-s) = (15,15-8+3-2) = (15,8)

The most important consideration in decimal division is to ensure that you are generating enough digits in the fractional part for the computation you are performing. Fractional digits are produced in two ways. First, the zero padding which occurs when the dividend is aligned provides 15-p fractional digits, so that dividend values with small precision generate more fractional digits. Second, if q is greater than s, then (q-s) additional fractional digits are generated as shown above. If, on the other hand, the dividend contains fewer fractional digits than the divisor then q is less than s, and (s-q) fractional digits are consumed. The simple case of q = s occurs quite often. In this particular situation, the number of fractional digits depends entirely upon the precision of the divisor, and results in 15-p fractional digits.

You may also wish to truncate or extend the result with zeroes using the DIVIDE built-in function during a particular computation (see the PL/I-80 Language Manual). The form is

DIVIDE (x,y,p,q)

where p and q are literal constants, can appear as an expression or subexpression in an arithmetic computation, and has the same effect as the statement

DECIMAL (x/y,p,q)

As above, the value x is divided by y, but the precision and scale values are forced (p,q). Note that the computation is carried out as described above, and the resulting value is then shifted by the appropriate number of digits in order to obtain the desired precision and scale.

12.6. Conversion Between Fixed Decimal and Float Binary.

It is often useful to convert Fixed Decimal values to and from a Float Binary representation. In PL/I-80, this conversion is accomplished by first converting to character format, then to either Fixed Decimal or Float Binary. Although conversion from Fixed Decimal, then to Character, and finally to Float Binary is provided directly in the language, a special library function, call "ftc," is provided for conversion from Float Binary to Character format. This particular function is useful in other applications, and is described fully in this section.

Consider the following program as an example of conversion between data formats:

```
conv:
    proc options(main);
    dcl
        ftc entry (float)
            returns (char(17) var);
    dcl
        d fixed decimal(8,2),
        f float binary;
d = -123456.78;
f = char(c);
f = 0.314159265e1;
d = ftc(f);
end conv;
```

In this example, the Fixed Decimal value d is first initialized to 123456.78. Next, the CHAR built-in function is applied to the Fixed Decimal value to produce a character string constant

'b-123456.78'

where "b" is a blank character. (Recall from the PL/I-80 Language Manual that conversion from Fixed Decimal to character produces a string of length p+3, consisting of leading blanks, a sign position, and digits of the number itself.) The store operation following the character conversion effectively converts from Fixed Decimal to Float Binary with possible truncation errors due to conversion to binary, as discussed previously. Next, the value of Pi is stored into the Float Binary value f. Normally, an assignment from f into d causes truncation of the fractional part, since the PL/I standard first requires conversion to Fixed Binary. Instead, the ftc function is applied to f to produce the character string variable of length 17:

'b3.14159200000000'

where the blank character, represented by b, is inserted if the number is positive, and "-" is included if the value is negative. The subsequent store operation into d produces a truncated value of 3.14, due to d's declared scale value of two decimal places. It should be noted that Float Binary representation allows approximately 7-1/2 significant decimal digits, and thus truncation errors may occur as

the conversions take place.

Additional examples of conversion between Fixed Decimal and Float Binary are given in the programs described below.

12.7. A Simple Loan Payment Schedule.

The first example of commercial processing is found in Figure 12-1. This program computes a loan payment schedule using three input values corresponding to the loan principal (P), the yearly interest rate (i), and monthly payment (PMT). Each month, the remaining principal is computed as

```
P + i * P
```

and is then reduced by the payment amount, producing a new principal for the next month:

```
P = (P + i * P) - PMT
```

The program iterates through the statements from line 18 through line 31 until the principal is reduced to zero, and the loan is completely paid off.

We assume in this program that the principal does not exceed $999,999,999.99, and thus the declaration on line 6 defines P as a Fixed Decimal variable with precision 11 and scale 2. Further, we shall assume that the payment does not exceed $9,999.99, so PMT is declared with precision 6 and scale 2. Finally, the interest rate is defined with the Fixed Decimal(4,2) attribute allowing numbers as large as 99.99%. The two variables "m" and "y" correspond to the month and year, beginning at the first month of the first year.

The initial values are read between lines 10 and 15. Note that for this example, no range checking is performed and thus negative values are acceptable, and payment values can be processed which would never pay off the loan. These checks must be made, of course, to be useful in an application environment.

On each monthly iteration, the month is incremented with possible overflow past the 12th month which changes the year value (lines 19 through 24). The current principal P is displayed on line 25, and the monthly interest is added on the following line. The computation on line 26 is evaluated as follows:

```
 1 a 0000 pmt:
 2 a 0006      proc options(main);
 3 c 0006      dcl
 4 c 0006          m   fixed binary,
 5 c 0006          y   fixed binary,
 6 c 0006          P   fixed decimal(11,2),
 7 c 0006          PMT fixed decimal(6,2),
 8 c 0006          i   fixed decimal(4,2);
 9 c 0006         do while('1'b);
10 c 0006          put skip list('Principal ');
11 c 0022          get list(P);
12 c 0041          put list('Interest  ');
13 c 0058          get list(i);
14 c 0077          put list('Payment   ');
15 c 008E          get list(PMT);
16 c 00AD          m = 0;
17 c 00B3          y = 0;
18 c 00B6              do while (P > 0);
19 c 00CC              if mod(m,12) = 0 then
20 c 00DF                  do;
21 c 00DF                  y = y + 1;
22 c 00E6                  put skip list('Year',y);
23 c 010D                  end;
24 c 010D              m = m + 1;
25 c 0114              put skip list(m,P);
26 c 0142              P = P + round( i * P / 1200, 2);
27 c 0182              if P < PMT then
28 c 0198                  PMT = P;
29 c 01A8              put list(PMT);
30 c 01C6              P = P - PMT;
31 c 01E7              end;
32 c 01E7          end;
33 a 01E7      end pmt;
```

Figure 12.1. Simple Loan Payment Program Part A.

```
B>pmta

Principal 500
Interest  14
Payment   22.10

Year        1
          1          500.00      22.10
          2          483.73      22.10
          3          467.27      22.10
          4          450.62      22.10
          5          433.78      22.10
          6          416.74      22.10
          7          399.50      22.10
          8          382.06      22.10
          9          364.42      22.10
         10          346.57      22.10
         11          328.51      22.10
         12          310.24      22.10
Year        2
         13          291.76      22.10
         14          273.06      22.10
         15          254.15      22.10
         16          235.02      22.10
         17          215.66      22.10
         18          196.08      22.10
         19          176.27      22.10
         20          156.23      22.10
         21          135.95      22.10
         22          115.44      22.10
         23           94.69      22.10
         24           73.69      22.10
Year        3
         25           52.45      22.10
         26           30.96      22.10
         27            9.22       9.33
Principal ^C
```

Figure 12.1. Simple Loan Payment Program Part B.

i	has precision and scale (4,2)
P	has precision and scale (11,2)
i * P	results in Fixed Decimal(15,4)
1200	has precision and scale (4,0)
(i * P)/1200	has precision (15,4), since precision and scale in division is computed as (15,15-15+4-0)

The division by 1200 is required since the interest rate is expressed as a percentage (division by 100) over a one year period (division by 12). The intermediate result is ROUNDed in the second decimal place (cents position), and added to the principal. This result becomes the new principal.

In the last month of payment, it is likely that the remaining principal is less than the payment. The test on line 27 accounts for this possibility and, if so, changes the payment to equal the principal on line 28. The payment is printed on line 29 and, finally, the principal is reduced by the payment on line 30 using the assignment

```
P = P - PMT
```

The output from this program is shown following the program listing in Figure 12-1, with an initial loan of $500, interest rate 14%, and payment of $22.10 per month.

12.8. Ordinary Annuity.

Given the interest rate (i) and two of three values, the annuity program listed in Figure 12-2 computes either the present value (PV), payment (PMT), or number of periods (n). This particular program illustrates the use of several commercial processing facilities of PL/I-80, including a mix of Floating Point and Fixed Decimal arithmetic, along with picture format output.

Unlike the program of the previous section, the annuity program computes the unknown value through static formulas, rather than iteration. The static formulas are given below, assuming the interest rate is greater than zero. First, the present value is given by:

$$PV = PMT \frac{1 - \frac{1}{(1+i)^n}}{i}$$

and, by transposing the above formula, PMT can be computed as

```
 1 a 0000 annuity:
 2 a 0006     proc options(main);
 3 c 0006     %replace
 4 c 000D         clear by '^z',
 5 c 000D         true  by '1'b;
 6 c 000D     dcl
 7 c 000D         PMT fixed decimal(7,2),
 8 c 000D         PV  fixed decimal(9,2),
 9 c 000D         IP  fixed decimal(6,6),
10 c 000D         x   float binary,
11 c 000D         yi  float binary,
12 c 000D         i   float binary,
13 c 000D         n   fixed;
14 c 000D     dcl
15 c 000D         ftc entry (float binary) returns (char(17) var);
16 c 000D
17 c 000D     put list (clear,'^i^iO R D I N A R Y   A N N U I T Y');
18 c 002F     put skip (2) list
19 c 004B         ('^iEnter Known Values, or 0, on Each Iteration');
20 c 004B
21 c 004B     on error
22 d 0052         begin;
23 e 0055         put skip list('^iInvalid Data, Re-enter');
24 e 0071         go to retry;
25 d 0074         end;
26 d 0074
27 c 0074     retry:
28 c 007B         do while (true);
29 c 007B         put skip(3) list
30 c 0097             ('^iPresent Value ');
31 c 0097         get list(PV);
32 c 00B6         put list('^iPayment       ');
33 c 00CD         get list(PMT);
34 c 00EC         put list('^iInterest Rate ');
35 c 0103         get list(yi);
36 c 011E         i = yi / 1200;
37 c 012F         put list('^iPay Periods   ');
38 c 0146         get list(n);
39 c 015E
40 c 015E         if PV = 0 | PMT = 0 then
41 c 0190             x = 1 - 1/(1+i)**n;
42 c 01B3
43 c 01B3         if PV = 0 then
44 c 01C9             do;
45 c 01C9             /* compute present value */
46 c 01C9             PV = PMT * dec(ftc(x/i),15,6);
47 c 01FD             put edit('^iPresent Value is ',PV)
48 c 022C                 (a,p'$$$,$$$,$$$V.99');
49 c 022C             end;
50 c 022C
51 c 022C         if PMT = 0 then
52 c 0242             do;
53 c 0242             /* compute payment */
54 c 0242             PMT = PV * dec(ftc(i/x),15,8);
55 c 0276             put edit('^iPayment is ',PMT)
```

Figure 12.2. Ordinary Annuity Program Part A.

```
56 c 02A5                   (a,p'$$,$$$,$$$V.99');
57 c 02A5               end;
58 c 02A5
59 c 02A5           if n = 0 then
60 c 02AE               do;
61 c 02AE               /* compute number of periods */
62 c 02AE               IP = ftc(i);
63 c 02C1               x = char(PV * IP / PMT);
64 c 02EF               n = ceil ( - log(1-x)/log(1+i) );
65 c 032C               put edit('^i',n,' Pay Periods')
66 c 0362                   (a,p'ZZZ9',a);
67 c 0362               end;
68 c 0362           end;
69 a 0362       end annuity;
```

```
                O R D I N A R Y   A N N U I T Y

        Enter Known Values, or 0, on Each Iteration

        Present Value 32000
        Payment       0
        Interest Rate 8.75
        Pay Periods   360
        Payment is       $251.74


        Present Value ,
        Payment       0
        Interest Rate ,
        Pay Periods   240
        Payment is       $282.78


        Present Value 0
        Payment       ,
        Interest Rate ,
        Pay Periods   ,
        Present Value is     $31,998.87


        Present Value 32000
        Payment       ,
        Interest Rate ,
        Pay Periods   0
         240 Pay Periods


        Present Value ^C
```

Figure 12.2. Ordinary Annuity Program Part B.

$$PMT = PV \frac{i}{1 - \frac{1}{(1 + i)^n}}$$

Finally, n is evaluated using:

$$n = - \frac{\text{Log}\left(1 - PV\left(\frac{i}{PMT}\right)\right)}{\text{Log}(1 + i)}$$

The program contains one main loop between lines 28 and 67 where the present value, payment, and yearly interest are read from the console. The operator must enter two non-zero values and one zero value on each iteration. The program then computes the value of the variable which was entered as zero. The values are retained on each main loop so that a comma (,) entry can be entered if the value is not to be changed. The interest rate, expressed as a yearly percentage, is reduced to a monthly period on line 36, where it is divided by 12 * 100 = 1200. Again, the program does not check for input values in the proper range. The interaction with the annuity program is shown following the program listing, with several different values used as input.

This particular program uses both Float Binary and Fixed Decimal computations since there is a mixture of simple decimal arithmetic and analytic functions. The variables used throughout the program are defined between lines 7 and 13 as follows. PMT holds the payment value, and is defined as a Fixed Decimal number as large as $99,999.99. Similarly, the present value can be as large as $99,999,999.99. The variable IP is used to hold the interest rate for a one month period, represented as a Decimal fraction with six decimal places. The variables x, yi and i are Float Binary numbers which are used during the computations to approximate decimal numbers with about 7-1/2 decimal places. Finally, the Fixed Binary variable n holds the number of payment periods, ranging from 1 to 32767.

Referring to the above formulas, the computation

```
1 - 1 / ( 1 + i ) ** n
```

occurs in both the computation of PV and PMT. Thus, line 41 stores this value into the variable x for subsequent use if either PV or PMT is to be evaluated. Again, it is important to realize that x is only an approximation to the decimal value given by this expression. If the operator enters a zero value for PV, then the statements between lines 45 and 49 are executed. In this case, PV is computed using the "ftc" external subroutine, defined on line 15, as

```
PV = PMT * dec(ftc(x/i),15,6)
```

where x/i is a Float Binary computation, and ftc converts the resulting value from float to character form. Given that x/i produces the value 3.042455E+01, for example, ftc(x/i) results in 30.42455 which is acceptable for conversion to decimal. The ERROR(1) condition is signalled by ftc, indicating a conversion error, if the floating point argument cannot be converted to a 15-digit decimal number. The "dec" function is applied to the character string to convert to a specific precision (15) and scale (6) for the subsequent multiplication. How did we decide on this particular value for precision and scale? First, consider a simpler form of this program which is shown below

```
dcl
    PMT fixed decimal(7,2),
    PV  fixed decimal(9,2),
    Q   fixed decimal(u,v);
PV = PMT * Q;
```

where we must decide upon the appropriate constant values for u and v. PV has precision and scale (9,2) and thus there must be 7 digits in the whole part and 2 digits in the fraction. We will generate the full 7 digits in the whole part if the product PMT * Q results in any of the following precision and scale values

(9,2) (10,3) (11,4) (12,5) (13,6) (14,7) (15,8)

since the assignment to PV will truncate any fractional digits beyond the second decimal place. Further, since PMT has precision and scale (7,2), we can choose (15,6) as the precision and scale of Q to produce

(min(15,7+15+1),2+6) = (15,8)

as the precision and scale resulting from the rules for multiplication stated previously. In general, given an expression with precision and scale values as shown below

```
  a     =    b     *    c
(p,q)      (r,s)      (u,v)
```

where p, q, r, and s are constants, you can set the precision and scale of c to

u = 15 v = 15 - p + q - s

which, using the values in the above statement, results in

v = 15 - 9 + 2 - 2 = 8, or (u,v) = (15,6)

as the precision and scale of Q.

Returning to the sample program in Figure 12-2, the resulting present value PV is written using a picture format with a drifting dollar sign on line 48.

Alternatively, the operator could have entered a non-zero

present value with a zero value for the payment (PMT). In this case, the group beginning at line 57 is entered, and the value of PMT is computed:

```
PMT = PV * dec (ftc(i/x),15,8);
```

using essentially the same technique as shown in the previous computation. Again, we must decide the precision and scale of the second operand in the multiplication. (We are really concerned only with the value of the scale since the precision can be taken as 15.) Using the analysis shown above, the form is

```
   a     =     b     *     c
(7,2)       (9,2)       (15,v)
```

where

```
v = 15 - p + q - s = 15 - 7 + 2 - 2 = 8
```

The computed value of PMT is written with the a picture format on line 56.

The final case occurs when the operator enters non-zero values for PV and PMT, but sets the number of periods to zero. When this occurs, the group beginning on line 60 is executed to compute n. First, the interest for a monthly period is changed from Float Binary to Fixed Decimal using the assignment on line 62. The next assignment

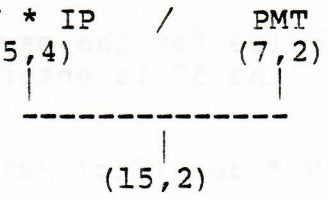
```
x = char (PV * IP / PMT)
```

first computes the partial Decimal result PV * IP / PMT, then converts the result to character, and then to Float Binary through the assignment to x. The intermediate character form is necessary since otherwise the intermediate result would first be converted to Fixed Binary, then to Float Binary, resulting in truncation of the fraction. (This sequence of conversions is necessary to maintain compatibility with the full language.)

First, we'll analyze the precision and scale of the Decimal computation. The subexpression PV * IP produces the following:

```
   PV      *      IP
 (9,2)          (7,2)
   |              |
   ----------------
          |
       (15,4)
```

The computation proceeds with the division, producing the following precision and scale:



since, according to the precision and scale rules for division,

(15,15-p+q-s) = (15,15-15+4-2) = (15,2)

thus providing two decimal places in the computation. Additional fractional digits can be generated by applying the decimal function following the multiply, as shown below

x = char(dec(PV*P, 11,4) / PMT)

which would produce a quotient with precision and scale

(15,15-11+4-2) = (15,6)

The resulting value, x, is used in the expression on line 64 to compute the number of payment periods. The CEIL function is applied to the result so that any partial month becomes a full month in the payment period analysis. The number of months is written using a picture format with leading zero suppression, and the program loops for another set of input values.

12.9. Formatted Loan Payment Schedule.

The program shown in Figure 12-3 is essentially the same as that presented in Section 12.7, with a more elaborate analysis and display format. As shown starting on line 116, this program reads several data items:

PV	Present Value (Initial Principal)
yi	Yearly Interest Rate
PMV	Monthly Payment
ir	Yearly Inflation Rate
sm	Starting Month of Payment (1-12)
sy	Starting Year of Payment (0-99)
fm	Fiscal Month (End of Fiscal Year, 1-12)
dl	Display Level (0-2)

The initial principal and payment variables are declared as Fixed Decimal (10,2) on lines 16 and 19, allowing values as large as $99,999,999.99. The yearly interest rate and yearly inflation rate are expressed as percentages as large as 99.99, as defined on lines 24 and 29. The month and year variables, sm, sy, and fm are in Fixed Binary format, and are assumed to properly represent month and year values. The variable dl defines the amount of information displayed

```
 1 a 0000 pmt:
 2 a 0006     proc options(main);
 3 c 0006     %replace
 4 c 000D         true   by '1'b,
 5 c 000D         false  by '0'b,
 6 c 000D         clear  by '^z';
 7 c 000D     dcl
 8 c 000D         end bit(1),
 9 c 000D         m   fixed binary,
10 c 000D         sm  fixed binary,
11 c 000D         y   fixed binary,
12 c 000D         sy  fixed binary,
13 c 000D         fm  fixed binary,
14 c 000D         dl  fixed binary,
15 c 000D         P   fixed decimal(10,2),
16 c 000D         PV  fixed decimal(10,2),
17 c 000D         PP  fixed decimal(10,2),
18 c 000D         PL  fixed decimal(10,2),
19 c 000D         PMT fixed decimal(10,2),
20 c 000D         PMV fixed decimal(10,2),
21 c 000D         INT fixed decimal(10,2),
22 c 000D         YIN fixed decimal(10,2),
23 c 000D         IP  fixed decimal(10,2),
24 c 000D         yi  fixed decimal(4,2),
25 c 000D         i   fixed decimal(4,2),
26 c 000D         INF fixed decimal(4,3),
27 c 000D         ci  fixed decimal(15,14),
28 c 000D         fi  fixed decimal(7,5),
29 c 000D         ir  fixed decimal(4,2);
30 c 000D
31 c 000D     dcl
32 c 000D         name char(14) var static init('$con'),
33 c 000D         output file;
34 c 000D
35 c 000D     put list(clear,'^i^iS U M M A R Y   O F   P A Y M E N T S');
36 c 002F
37 c 002F     on undefinedfile(output)
38 d 0037         begin;
39 e 003A         put skip list('^i^icannot write to',name);
40 e 005F         go to open_output;
41 d 0062         end;
42 d 0062
43 c 0062     open_output:
44 c 0069         put skip(2) list('^i^iOutput File Name ');
45 c 0085         get list(name);
46 c 009F
47 c 009F     if name = '$con' then
48 c 00AD         open file(output) title('$con') print pagesize(0);
49 c 00CC     else
50 c 00CC         open file(output) title(name) print;
51 c 00E6
52 c 00E6     on error
53 d 00ED         begin;
54 e 00F0         put skip list('^i^iBad Input Data, Retry');
55 e 010C         go to retry;
```

Figure 12.3. Summary of Loan Payments Program Part A.

```
 56 d 010F          end;
 57 d 010F
 58 c 010F      retry:
 59 c 0116          do while(true);
 60 c 0116          put skip(2)
 61 c 0132              list('^i^iPrincipal      ');
 62 c 0132          get list(PV);
 63 c 0151          P = PV;
 64 c 0161          put list('^i^iInterest       ');
 65 c 0178          get list(yi);
 66 c 0197          i = yi;
 67 c 01A7          put list('^i^iPayment        ');
 68 c 01BE          get list(PMV);
 69 c 01DD          PMT = PMV;
 70 c 01ED          put list('^i^i%Inflation     ');
 71 c 0204          get list(ir);
 72 c 0223          fi = 1 + ir/1200;
 73 c 0253          ci = 1.00;
 74 c 0263          put list('^i^iStarting Month ');
 75 c 027A          get list(sm);
 76 c 0292          put list('^i^iStarting Year  ');
 77 c 02A9          get list(sy);
 78 c 02C1          put list('^i^iFiscal Month   ');
 79 c 02D8          get list(fm);
 80 c 02F0          put edit('^i^iDisplay Level  ',
 81 c 032E               '^i^iYr Results : 0 ',
 82 c 032E               '^i^iYr Interest: 1 ',
 83 c 032E               '^i^iAll Values : 2 ')
 84 c 032E              (skip,a);
 85 c 032E          get list(dl);
 86 c 0346          if dl < 0 | dl > 2 then
 87 c 0357              signal error;
 88 c 035E          m = sm;
 89 c 0364          y = sy;
 90 c 036A          IP = 0;
 91 c 037A          PP = 0;
 92 c 038A          YIN = 0;
 93 c 039A          if name ^= '$con' then
 94 c 03A8              put file(output) page;
 95 c 03BA          call header();
 96 c 03BD              do while (P > 0);
 97 c 03D3              end = false;
 98 c 03D8              INT = round ( i * P / 1200, 2 );
 99 c 0408              IP  = IP + INT;
100 c 0423              PL  = P;
101 c 0433              P   = P  + INT;
102 c 044E              if P < PMT then
103 c 0464                  PMT = P;
104 c 0474              P   = P - PMT;
105 c 048F              PP  = PP + (PL - P);
106 c 04B5              INF = ci;
107 c 04CA              ci = ci / fi;
108 c 04EA              if P = 0 | dl > 1 | m = fm then
109 c 0520                  do;
110 c 0520                  put file(output) skip
```

Figure 12.3. Summary of Loan Payments Program Part B.

```
111 c 055B                      edit('|',100*m+y) (a,p'99/99');
112 c 055B                  call display(PL * INF, INT * INF,
113 c 0601                  PMT * INF, PP * INF, IP * INF);
114 c 0601                  end;
115 c 0601              if m = fm & dl > 0 then
116 c 061E                  call summary();
117 c 0621              m = m + 1;
118 c 0628              if m > 12 then
119 c 0634                  do;
120 c 0634                  m = 1;
121 c 063A                  y = y + 1;
122 c 0641                  if y > 99 then
123 c 064D                      y = 0;
124 c 0656                  end;
125 c 0656              end;
126 c 0656          if dl = 0 then
127 c 065F              call line();
128 c 0665          else
129 c 0665          if ^end then
130 c 066C              call summary();
131 c 0672          end;
132 c 0672
133 c 0672      display:
134 c 0672          proc(a,b,c,d,e);
135 e 0672          dcl
136 e 067F              (a,b,c,d,e) fixed decimal(10,2);
137 e 067F          put file (output) edit
138 e 0731              ('|',a,'|',b,'|',c,'|',d,'|',e,'|')
139 e 0731              (a,2(2(p'$zz,zzz,zz9v.99',a),
140 e 0731                      p'$zzz,zz9.v99',a));
141 c 0731          end display;
142 c 0731
143 c 0731      summary:
144 c 0731          proc;
145 e 0731          end = true;
146 e 0736          call current_year(IP-YIN);
147 e 0757          YIN = IP;
148 c 0768          end summary;
149 c 0768
150 c 0768      current_year:
151 c 0768          proc(I);
152 e 0768          dcl
153 e 076F              yp fixed binary,
154 e 076F              I fixed decimal(10,2);
155 e 076F          yp = y;
156 e 0775          if fm < 12 then
157 e 0781              yp = yp - 1;
158 e 0788          call line();
159 e 078B          put skip file(output) edit
160 e 0804          ('|','Interest Paid During ''',yp,'-''',y,' is ',I,'|')
161 e 0804          (a,x(15),2(a,p'99'),a,p'$$$,$$$,$$9V.99',x(16),a);
162 e 0804          call line();
163 c 0808          end current_year;
164 c 0808
165 c 0808      header:
```

Figure 12.3. Summary of Loan Payments Program Part C.

```
166 c 0808          proc;
167 e 0808          put file(output) list(clear);
168 e 0822          call line();
169 e 0825          put file(output) skip edit
170 e 0860          ('|','L O A N   P A Y M E N T   S U M M A R Y','|')
171 e 0860              (a,x(19));
172 e 0860          call line();
173 e 0863          put file(output) skip edit
174 e 08E3          ('|','Interest Rate',yi,'%','Inflation Rate',ir,'%','|')
175 e 08E3              (a,x(15),2(a,p'b99v.99',a,x(6)),x(9),a);
176 e 08E3          call line();
177 e 08E6          put file(output) skip edit
178 e 0942              ('|Date |',
179 e 0942               '  Principal   |',
180 e 0942               'Plus  Interest|',
181 e 0942               '  Payment   |',
182 e 0942               'Principal Paid|',
183 e 0942               'Interest Paid |') (a);
184 e 0942          call line();
185 c 0946          end header;
186 c 0946
187 c 0946      line:
188 c 0946          proc;
189 e 0946          dcl
190 e 0946              i fixed bin;
191 e 0946          put file(output) skip edit
192 e 099E          ('-------','------------',
193 e 099E              ('---------------' do i = 1 to 4)) (a);
194 c 099E          end line;
195 a 099E      end pmt;
```

```
                S U M M A R Y   O F   P A Y M E N T S

                Output File Name ,


                Principal       3000
                Interest        14
                Payment         144.03
                %Inflation      0
                Starting Month  11
                Starting Year   80
                Fiscal Month    12

                Display Level
                Yr Results : 0
                Yr Interest: 1
                All Values : 2 0
```

Figure 12.3. Summary of Loan Payments Program Part D.

LOAN PAYMENT SUMMARY					
Interest Rate 14.00%			Inflation Rate 00.00%		
Date	Principal	Plus Interest	Payment	Principal Paid	Interest Paid
12/80	$ 2,890.97	$ 33.73	$ 144.03	$ 219.33	$ 68.73
12/81	$ 1,479.02	$ 17.26	$ 144.03	$ 1,647.75	$ 368.67
11/82	$ 0.25	$ 0.00	$ 0.25	$ 3,000.00	$ 456.97

```
Principal      ,
Interest       ,
Payment        ,
%Inflation     ,
Starting Month ,
Starting Year  ,
Fiscal Month   ,

Display Level
Yr Results : 0
Yr Interest: 1
All Values : 2 1
```

LOAN PAYMENT SUMMARY					
Interest Rate 14.00%			Inflation Rate 00.00%		
Date	Principal	Plus Interest	Payment	Principal Paid	Interest Paid
12/80	$ 2,890.97	$ 33.73	$ 144.03	$ 219.33	$ 68.73
Interest Paid During '80-'80 is				$68.73	
12/81	$ 1,479.02	$ 17.26	$ 144.03	$ 1,647.75	$ 368.67
Interest Paid During '81-'81 is				$299.94	
11/82	$ 0.25	$ 0.00	$ 0.25	$ 3,000.00	$ 456.97
Interest Paid During '82-'82 is				$88.30	

Figure 12.3. Summary of Loan Payments Program Part E.

```
Principal       ,
Interest        ,
Payment         ,
%Inflation      ,
Starting Month  ,
Starting Year   ,
Fiscal Month    ,

Display Level
Yr Results : 0
Yr Interest: 1
All Values : 2 2
```

L O A N P A Y M E N T S U M M A R Y					
Interest Rate 14.00%			Inflation Rate 00.00%		
Date	Principal	Plus Interest	Payment	Principal Paid	Interest Paid
11/80	$ 3,000.00	$ 35.00	$ 144.03	$ 109.03	$ 35.00
12/80	$ 2,890.97	$ 33.73	$ 144.03	$ 219.33	$ 68.73
Interest Paid During '80-'80 is				$68.73	
01/81	$ 2,780.67	$ 32.44	$ 144.03	$ 330.92	$ 101.17
02/81	$ 2,669.08	$ 31.14	$ 144.03	$ 443.81	$ 132.31
03/81	$ 2,556.19	$ 29.82	$ 144.03	$ 558.02	$ 162.13
04/81	$ 2,441.98	$ 28.49	$ 144.03	$ 673.56	$ 190.62
05/81	$ 2,326.44	$ 27.14	$ 144.03	$ 790.45	$ 217.76
06/81	$ 2,209.55	$ 25.78	$ 144.03	$ 908.70	$ 243.54
07/81	$ 2,091.30	$ 24.40	$ 144.03	$ 1,028.33	$ 267.94
08/81	$ 1,971.67	$ 23.00	$ 144.03	$ 1,149.36	$ 290.94
09/81	$ 1,850.64	$ 21.59	$ 144.03	$ 1,271.80	$ 312.53
10/81	$ 1,728.20	$ 20.16	$ 144.03	$ 1,395.67	$ 332.69
11/81	$ 1,604.33	$ 18.72	$ 144.03	$ 1,520.98	$ 351.41
12/81	$ 1,479.02	$ 17.26	$ 144.03	$ 1,647.75	$ 368.67
Interest Paid During '81-'81 is				$299.94	
01/82	$ 1,352.25	$ 15.78	$ 144.03	$ 1,776.00	$ 384.45
02/82	$ 1,224.00	$ 14.28	$ 144.03	$ 1,905.75	$ 398.73
03/82	$ 1,094.25	$ 12.77	$ 144.03	$ 2,037.01	$ 411.50
04/82	$ 962.99	$ 11.23	$ 144.03	$ 2,169.81	$ 422.73
05/82	$ 830.19	$ 9.69	$ 144.03	$ 2,304.15	$ 432.42
06/82	$ 695.85	$ 8.12	$ 144.03	$ 2,440.06	$ 440.54
07/82	$ 559.94	$ 6.53	$ 144.03	$ 2,577.56	$ 447.07
08/82	$ 422.44	$ 4.93	$ 144.03	$ 2,716.66	$ 452.00
09/82	$ 283.34	$ 3.31	$ 144.03	$ 2,857.38	$ 455.31
10/82	$ 142.62	$ 1.66	$ 144.03	$ 2,999.75	$ 456.97
11/82	$ 0.25	$ 0.00	$ 0.25	$ 3,000.00	$ 456.97
Interest Paid During '82-'82 is				$88.30	

Figure 12.3. Summary of Loan Payments Program Part F.

```
Principal       ,
Interest        ,
Payment         ,
%Inflation      10
Starting Month  ,
Starting Year   ,
Fiscal Month    10

Display Level
Yr Results : 0
Yr Interest: 1
All Values : 2 2
```

L O A N P A Y M E N T S U M M A R Y

Interest Rate 14.00% Inflation Rate 10.00%

Date	Principal	Plus Interest	Payment	Principal Paid	Interest Paid
11/80	$ 3,000.00	$ 35.00	$ 144.03	$ 109.03	$ 35.00
12/80	$ 2,864.95	$ 33.42	$ 142.73	$ 217.35	$ 68.11
01/81	$ 2,733.39	$ 31.88	$ 141.58	$ 325.29	$ 99.45
02/81	$ 2,602.35	$ 30.36	$ 140.42	$ 432.71	$ 129.00
03/81	$ 2,471.83	$ 28.83	$ 139.27	$ 539.60	$ 156.77
04/81	$ 2,341.85	$ 27.32	$ 138.12	$ 645.94	$ 182.80
05/81	$ 2,212.44	$ 25.81	$ 136.97	$ 751.71	$ 207.08
06/81	$ 2,083.60	$ 24.31	$ 135.82	$ 856.90	$ 229.65
07/81	$ 1,955.36	$ 22.81	$ 134.66	$ 961.48	$ 250.52
08/81	$ 1,829.70	$ 21.34	$ 133.65	$ 1,066.60	$ 269.99
09/81	$ 1,702.58	$ 19.86	$ 132.50	$ 1,170.05	$ 287.52
10/81	$ 1,576.11	$ 18.38	$ 131.35	$ 1,272.85	$ 303.41
Interest Paid During '80-'81 is				$332.69	
11/81	$ 1,451.91	$ 16.94	$ 130.34	$ 1,376.48	$ 318.02
12/81	$ 1,326.68	$ 15.48	$ 129.19	$ 1,478.03	$ 330.69
01/82	$ 1,203.50	$ 14.04	$ 128.18	$ 1,580.64	$ 342.16
02/82	$ 1,079.56	$ 12.59	$ 127.03	$ 1,680.87	$ 351.67
03/82	$ 957.46	$ 11.17	$ 126.02	$ 1,782.38	$ 360.06
04/82	$ 835.87	$ 9.74	$ 125.01	$ 1,883.39	$ 366.92
05/82	$ 714.79	$ 8.34	$ 124.00	$ 1,983.87	$ 372.31
06/82	$ 594.25	$ 6.93	$ 123.00	$ 2,083.81	$ 376.22
07/82	$ 474.26	$ 5.53	$ 121.99	$ 2,183.19	$ 378.66
08/82	$ 354.84	$ 4.14	$ 120.98	$ 2,281.99	$ 379.68
09/82	$ 236.02	$ 2.75	$ 119.97	$ 2,380.19	$ 379.27
10/82	$ 117.80	$ 1.37	$ 118.96	$ 2,477.79	$ 377.45
Interest Paid During '81-'82 is				$124.28	
11/82	$ 0.20	$ 0.00	$ 0.20	$ 2,457.00	$ 374.25
Interest Paid During '81-'82 is				$0.00	

Figure 12.3. Summary of Loan Payments Program Part G.

during a particular iteration of the program, where 0 provides an abbreviated display, 1 provides additional information, and 2 gives the full trace.

Using an algorithm similar to that described in Section 12.7, the primary loop occurs between lines 96 and 131, where the initial principal is increased by the monthly interest and reduced by the monthly payment until the principal becomes zero. Several examples of program interaction are shown following the listing of Figure 12-3. The first output listing shows a minimal display corresponding to a loan of $3000 at 14% interest rate with a payment of $144.03. In this case, an inflation rate of 0% is assumed with a starting payment on 11/80, and end-of-year taxes due in December of each year. The display shows the principal, interest in December, monthly payment, amount paid toward principal in December, and amount of interest paid in the last month of the fiscal year.

The second output listing shows an execution of the main loop using the same values shown above, with display level 1. In this case, the output also contains the yearly interest paid on the loan for each fiscal year which would, presumably, be deducted from the taxable income.

The third output listing again uses the same initial values used in the previous examples, but provides a full display of the monthly principal, interest, monthly payment, payment applied to the principal, and interest payment.

The last display shows the same loan and interest rate with an adjustment in dollar value due to inflation. The (rather conservative) inflation rate of 10% is assumed in this example, so that all amounts are scaled to the value of the dollar at the time the loan was issued. For tax reporting purposes, the display showing the total interest paid at the end of each year is not scaled, and thus does not match the sum of the interest paid during the year. It is interesting to note that if we assume a 0% inflation rate, the total loan payment is 3,456.97, taken from the previous output. Assuming an inflation rate of 10%, however, the total cost of the loan in today's dollars is

```
  2,457.00
+   374.25
--------
  2,831.25
```

resulting in a net gain of 68.75 over a two year period!

Several operational details must be presented in order to properly understand the operation of this program. First, there are several additional variables declared between lines 15 and 29 which are used throughout the program:

P	initially set to PV, but changes during execution (see lines 63, 101, and 104)
PP	total principal paid (see line 105)
PL	principal for current line, holds P for display purposes (see lines 100 and 112)
PMT	payment initially set to PMV, but changes during execution (see lines 69 and 103)
INT	computed interest during current month (see lines 98, 101, and 112)
YIN	interest at beginning of current year (see lines 92, 146, and 147)
IP	total interest paid (see lines 90, 99, and 146)
i	interest rate, initialized to yi (see line 66)
INF	percent of devaluation of the original dollar due to inflation (see lines 106, 112, and 113)
ci	current devaluation due to inflation (see lines 73, 106, and 107)
fi	factor for computing current inflation (see lines 72 and 107)

It should be noted that P and PMT are "working" variables for principal and payment so that the original variables PV and PMV are not destroyed during the computations. As a result, the operator can simply enter a comma (,) for subsequent input requests to indicate that the previously entered value is to be retained.

The program execution actually begins on line 35 with a "clear screen" character for the Lear-Siegler ADM-3A CRT. This control character is defined in the replace statement on line 6. If you are not using an ADM-3A, you can substitute the proper character in the replace statement and recompile the program.

In preparation for the subsequent OPEN, an ON-condition is set to trap possible OPEN errors (see lines 37 through 41). The operator is then prompted for the report output file name on line 44. The character variable "name" is initialized to the value "$con" on line 32: if the operator enters a comma rather than a file or device name, the console is assumed as the output device. If either a comma or the name $con is entered as the output file name, the console is OPENed with a zero page size so that no form-feeds are issued at the end of each logical page (see lines 47 and 48). Otherwise, the output file or device is OPENed as a normal PRINT device so that form-feeds are placed into the output file or sent to the physical output device

(usually the printer, $lst).

The ON-condition set at line 52 traps any occurrence of the ERROR condition, including ERROR(1) which indicates a data conversion error (a complete list of the ERROR subcodes is given in the "Recoverable Errors" section of the PL/I-80 Command Summary). Invalid data is also programmatically SIGNALed on line 87 if the value of dl is out-of-range. To make this particular program commercially palatable, it would be necessary to SIGNAL errors for all other invalid input data items, such as a negative interest rate. Further, the Fixed Overflow condition (FOFL) should also be set to intercept out-of-bound computations.

Program variable initialization for each set of input values begins on line 88. A page-eject is executed if the output file is not the console, followed by a page header printed by the "header" subroutine on line 165. It is instructional to compare the formatting statements in the header subroutine with the output values shown following the program listing.

The main processing loop, beginning at line 96, is executed repetitively until the principal reduces to zero. The variable "end" indicates whether or not an end-of-year summary has been printed (see line 145), and is used at the end of processing to avoid a possible duplicate summary (see line 129). The monthly interest (INT) for the current principal (P) is then computed and summed in IP on lines 98 and 99. The current principal is saved for later display in PL, and the monthly interest is added to the principal. If the payment exceeds the remaining principal on line 102, then the payment is reduced to cover this remainder. The principal is then reduced by the payment amount, which will eventually produce a zero value (if the original payment is sufficiently large to pay off the loan!). The total principal paid is summed on line 105, and the inflation rate is computed on line 106.

Since we have three display formats, the decision to display the current computation is somewhat complicated: if this is the last iteration (the principal P is zero), or if the full display format is selected (dl > 1), or if the current month is the end of the fiscal year (m = fm) then the current computation is written between lines 109 and 114. The picture format p'99/99' displays the month and year, where 100*m+y produces a four-digit number to match this format. If, for example, m = 11 and y = 64, then

```
100 * m + y   =   100 * 11 + 64   =   1164
```

which appears as 11/64 when printed using this picture. The "display" subroutine actually performs the output function, based upon the six actual parameters listed on lines 112 and 113. Each argument is adjusted by the current inflation rate INF and passed to the display subroutine. If the inflation rate has been set to 0%, the value of INF is 1.00 at this point in the computation. The body of the display subroutine, listed between lines 134 and 141 could, of course, be inserted in-line since there is only one call to display. However, the display subroutine does illustrate Fixed Decimal parameter passing

mechanisms and serves to break the program into smaller, more readable, segments. Again, it may be worthwhile comparing the formatting operations in the display subroutine with the actual program output.

The statement on line 115 then checks for the end of fiscal year (m = fm) and, if the display mode is either 1 or 2, a yearly interest summary is printed using the "summary" subroutine. The summary subroutine, listed between lines 144 and 148, in turn, calls the "current_year" subroutine to write the yearly interest paid (IP-YIN). The base value for next year's display is retained in YIN through the assignment on line 147. The current_year subroutine is listed between lines 151 and 163. If the fiscal year does not end in December (fm<12), the interest rate payment is split between two calendar years (yp = y - 1). Again, the current_year subroutine could be combined with the summary subroutine without changing the program logic.

The end of the main loop, between lines 126 and 130, contains statements which finalize the report. If the abbreviated display format was selected (dl = 0), a simple line of dashes completes the display. Otherwise, a check is made to ensure there have been intervening output lines (^end) and, if so, an interest summary is printed on line 130. The program then returns to the top of the loop and reads additional input parameters for production of another report.

12.10. Computation of Depreciation Schedules.

The final example illustrates a number of commercial processing concepts in PL/I-80 using evaluation of Depreciation Schedules as an example. The sample program listing is shown in Figure 12-4 followed by several examples of program interaction.

The Depreciation program reads several input values and prints a table based upon these values according to one of three different depreciation schedules: Straight-Line, Sum of the Years, or Double Declining. The program also accounts for bonus depreciation during the first year, reduction in taxable income due to sales tax, and investment tax credit on new or used equipment. The following general algorithms are used in this program:

> Investment Tax Credit (ITC) is assumed to be 10% of the selling price (see the replace statment, line 7), applied to the full price of new equipment, or up to $100,000 in the case of used equipment.
>
> Bonus Depreciation is assumed to be 10% of the selling price, up to a maximum of $2,000 (see the replace statement, lines 8 and 9).
>
> Under all three depreciation schedules, the amount to

depreciate is taken as the difference between the selling price minus the bonus depreciation, and the residual value of the equipment.

Under all schedules, the depreciation value computed for the first year is prorated by month through the remainder of the fiscal year (not including bonus depreciation).

In the case of Straight-Line depreciation, the amount to depreciate is spread uniformly over the number of years in which the depreciation occurs.

For the Sum of the Years, the year values are summed starting at 1, through the number of years in which depreciation takes place:

$$ys = 1 + 2 + 3 + \ldots + years$$

The depreciation is distributed over the total number of years by computing years/ys times the depreciation value for the first year, (years-1)/ys times the remainder for the second year, and so-forth until the last year in which 1/ys times the remaining depreciation value is taken.

For the Double Declining case, each year's depreciation is computed as the book value divided by the number of years, which is then multiplied by 2 for new equipment, or 1.5 if the equipment is used.

The program reads the selling price, residual value, percentage sales tax, the percentage income tax bracket, the number of months remaining in the current fiscal year, and the number of years in which to depreciate the equipment. The program then asks whether the equipment is new or used, and then reads the depreciation schedule code for the subsequent report. A sample input sequence is given in Figure 12-4, immediately following the program listing. Although the exact details of program organization and flow is left to the reader as an exercise, there are a number of constructs in this program worthy of discussion.

First, this particular program uses an entry variable array to "dispatch" the calls to compute one of three schedules. The entry array is defined on line 40, with a subscript range of 0 through 3. The individual elements of this vector are initialized between lines 42 and 45, allowing an indirect call to either the "error" subroutine or one of the depreciation schedule handling subroutines. The actual call to one of these subroutines occurs later in the program. The schedule selection takes place on line 71, where one of the characters s, y, or d is read from the console into the character variable "select_sched." After variable initialization has occurred, the "display" subroutine is invoked from line 89. The display subroutine, listed between lines 97 and 101, performs the actual dispatch to the schedule handler through the statement

```
call schedule (index (schedules,select_sched))
```

This particular statement can be decomposed as follows. The "schedules" variable is defined on line 39 and initialized to the character string 'syd', where each letter corresponds to one of the valid schedule handlers, as shown below

```
'syd'
 123
 |||
 ||---- double_declining
 |----- sum_of_years
 ------ straight_line
```

The evaluation of

```
index (schedules,select_sched)
```

is the same as

```
index('syd',select_sched)
```

which, for valid inputs s, y, or d, produces 1, 2, or 3. If the value of select_sched is not one of s, y, or d, then the index function returns a zero value. Thus, if select_sched is s, the call statement evaluates to

```
call schedule(1)
```

which, due to the assignment on line 43, calls the subroutine "straight_line." Similarly, an input of y or d produces

```
call schedule(2)  or  call schedule(3)
```

producing a call to "sum_of_years" or "double_declining," respectively. Since the index function returns zero if select_sched is not one of s, y, or d, all invalid character input values produce

```
call schedule(0)
```

which calls the "error" subroutine where the error condition is reported to the operator.

The second construct of interest in this program is the use of the "output" file variable, defined on line 35. During the parameter input phase, the operator is prompted with

```
List? (yes/no)
```

If the operator reponds with "yes" then the program writes the depreciation report to both the console and the listing device. The manner in which the program performs this function is presented below.

Two file constants, sysprint and list, are declared on line 36 to address the console and the list device. The console file is OPENed first, on line 47, using an infinite page length to avoid form-

feed characters. If, on any iteration of the main loop, the operator responds in the affirmative on line 73, the list device is subsequently OPENed on line 75. It should be noted that this statement may be executed several times on any particular execution of this program, but only the first OPEN has any effect. The "display" subroutine is called on line 89 to compute and display the output report for a specific set of input values. Display has a single actual parameter which is the file constant "sysprint" passed to the subroutine as the formal parameter "f" on line 99. The formal parameter, in turn, is assigned to the global variable "output" on line 100. Subsequent PUT statements of the form

```
put file(output) ...
```

write data to the console, producing the first report.

Referring to line 90 of Figure 12-4, if "copy_to_list" has the character value 'yes' then display is called once again. This time, however, the actual parameter is "list" which corresponds to the system listing device. Similar to the actions given above, the output file variable is indirectly assigned the value "list" and all PUT statements which reference file "output" write their data to the printer, resulting in both a soft and hard copy of the report.

Again, it is worthwhile examining the various components of this program while cross-checking output formats with the displayed results, since there are several different forms of decimal arithmetic and formatting which occur throughout.

```
 1 a 0000 depreciate:
 2 a 0006     procedure options(main);
 3 a 0006
 4 c 0006     %replace
 5 c 0006         clear_screen by '^z',
 6 c 0006         indent   by 15,
 7 c 0006         ITC_rate by .1,
 8 c 0006         bonus_rate by .1,
 9 c 0006         bonus_max  by 2000;
10 c 0006
11 c 0006     declare
12 c 0006         selling_price  decimal(8,2),
13 c 0006         adj_price decimal(8,2),
14 c 0006         residual_value decimal(8,2),
15 c 0006         year_value decimal(8,2),
16 c 0006         depreciation_value decimal(8,2),
17 c 0006         total_depreciation decimal(8,2),
18 c 0006         book_value decimal(8,2),
19 c 0006         tax_rate decimal(3,2),
20 c 0006         sales_tax decimal(8,2),
21 c 0006         tax_bracket decimal(2),
22 c 0006         FYD decimal(8,2),
23 c 0006         ITC decimal(8,2),
24 c 0006         bonus_dep decimal(8,2),
25 c 0006         months_remaining decimal(2),
26 c 0006         new char(4),
27 c 0006         factor decimal(2,1),
28 c 0006         years decimal(2),
29 c 0006         year_sum decimal(3),
30 c 0006         current_year decimal(2),
31 c 0006         select_sched char(1);
32 c 0006
33 c 0006     declare
34 c 0006         copy_to_list char(4),
35 c 0006         output file variable,
36 c 0006         (sysprint, list) file;
37 c 0006
38 c 0006     declare
39 c 0006         schedules char(3) static initial ('syd'),
40 c 0006         schedule (0:3) entry variable;
41 c 0006
42 c 0006     schedule (0) = error;
43 c 000C     schedule (1) = straight_line;
44 c 0015     schedule (2) = sum_of_years;
45 c 001E     schedule (3) = double_declining;
46 c 0027
47 c 0027     open file (sysprint) stream print pagesize(0)
48 c 0043         title ('$con');
49 c 0043
50 c 0043         do while('1'b);
51 c 0043         put list(clear_screen,'^i^i^iDepreciation Schedule');
52 c 0065         put skip(3) list('^i^iSelling  Price? ');
53 c 0081         get list(selling_price);
54 c 00A0         put list('^i^iResidual Value? ');
55 c 00B7         get list(residual_value);
```

Figure 12.4. Depreciation Schedule Program Part A.

```
 56 c 00D6          put list('^i^iSales Tax (%)?  ');
 57 c 00ED          get list(tax_rate);
 58 c 010C          put list('^i^iTax Bracket(%)? ');
 59 c 0123          get list(tax_bracket);
 60 c 0142          put list('^i^iProRate Months? ');
 61 c 0159          get list(months_remaining);
 62 c 0178          put list('^i^iHow Many Years? ');
 63 c 018F          get list(years);
 64 c 01AE          put list('^i^iNew? (yes/no)   ');
 65 c 01C5          get list(new);
 66 c 01DF          put edit('^i^iSchedule:',
 67 c 021D               '^i^iStraight    (s)',
 68 c 021D               '^i^iSum-of-Yrs (y)',
 69 c 021D               '^i^iDouble Dec (d)? ')
 70 c 021D              (a,skip);
 71 c 021D          get list(select_sched);
 72 c 0237          put list('^i^iList? (yes/no)  ');
 73 c 024E          get list(copy_to_list);
 74 c 0268          if copy_to_list = 'yes' then
 75 c 0278              open file(list) stream print title('$lst');
 76 c 0294          factor = 1.5;
 77 c 02A4          if new = 'yes' then
 78 c 02B4              factor = 2.0;
 79 c 02C4          sales_tax =
 80 c 0304              decimal(selling_price*tax_rate,12,2)/100+.005;
 81 c 0304          if new = 'yes' | selling_price <= 100000.00 then
 82 c 032E              ITC = selling_price * ITC_rate;
 83 c 0351          else
 84 c 0351              ITC = 100000 * ITC_rate;
 85 c 0371          bonus_dep = selling_price * bonus_rate;
 86 c 0391          if bonus_dep > bonus_max then
 87 c 03A7              bonus_dep = bonus_max;
 88 c 03B7          put list(clear_screen);
 89 c 03CE          call display(sysprint);
 90 c 03DA          if copy_to_list = 'yes' then
 91 c 03EA              call display(list);
 92 c 03F6          put skip list('^i^i^i   Type RETURN to Continue');
 93 c 0412          get skip(2);
 94 c 0426          end;
 95 c 0426
 96 c 0426 display:
 97 c 0426      procedure(f);
 98 e 0426      declare
 99 e 042D          f file;
100 e 042D      output = f;
101 e 0437      call schedule (index (schedules,select_sched));
102 c 0453      end display;
103 c 0453
104 c 0453 error:
105 c 0453      procedure;
106 c 0453      /* bad entry for schedule */
107 e 0453      put file (output) edit('Invalid Schedule - Enter s, y, or d')
108 e 0473          (page,column(indent),x(8),a);
109 e 0473      call line();
110 c 0477      end error;
```

Figure 12.4. Depreciation Schedule Program Part B.

```
111 c 0477
112 c 0477 straight_line:
113 c 0477     procedure;
114 e 0477     adj_price = selling_price - bonus_dep;
115 e 0492     put file (output) edit('S T R A I G H T   L I N E')
116 e 04B2         (page,column(indent),x(14),a);
117 e 04B2     call header();
118 e 04B5     depreciation_value = adj_price - residual_value;
119 e 04D0     book_value = adj_price;
120 e 04E0     total_depreciation = 0;
121 e 04F0         do current_year = 1 to years;
122 e 0526         year_value =
123 e 0560             decimal(depreciation_value/years,8,2) + .005;
124 e 0560         if current_year = 1 then
125 e 0576             do;
126 e 0576             year_value =
127 e 05A6                 year_value * months_remaining / 12;
128 e 05A6             FYD = year_value;
129 e 05B6             end;
130 e 05B6         depreciation_value = depreciation_value - year_value;
131 e 05D1         total_depreciation = total_depreciation + year_value;
132 e 05EC         book_value = adj_price - total_depreciation;
133 e 0607         call print_line();
134 e 0624         end;
135 e 0624     call summary();
136 c 0628     end straight_line;
137 c 0628
138 c 0628 sum_of_years:
139 c 0628     procedure;
140 e 0628     adj_price = selling_price - bonus_dep;
141 e 0643     put file (output) edit('S U M   O F   T H E   Y E A R S')
142 e 0663         (page,column(indent),x(11),a);
143 e 0663     call header();
144 e 0666     depreciation_value = adj_price - residual_value;
145 e 0681     book_value = adj_price;
146 e 0691     total_depreciation = 0;
147 e 06A1     year_sum = 0;
148 e 06B1         do current_year = 1 to years;
149 e 06E7         year_sum = year_sum + current_year;
150 e 071C         end;
151 e 071C
152 e 071C         do current_year = 1 to years;
153 e 0752         year_value =
154 e 07A3             decimal(depreciation_value *
155 e 07A3                 (years - current_year + 1),12,2)
156 e 07A3                 / year_sum + .005;
157 e 07A3         if current_year = 1 then
158 e 07B9             do;
159 e 07B9             year_value =
160 e 07E9                 year_value * months_remaining / 12;
161 e 07E9             FYD = year_value;
162 e 07F9             end;
163 e 07F9         depreciation_value = depreciation_value - year_value;
164 e 0814         total_depreciation = total_depreciation + year_value;
165 e 082F         book_value = adj_price - total_depreciation;
```

Figure 12.4. Depreciation Schedule Program Part C.

```
166 e 084A          call print_line();
167 e 0867          end;
168 e 0867      call summary();
169 c 086B      end sum_of_years;
170 c 086B
171 c 086B double_declining:
172 c 086B      procedure;
173 e 086B      adj_price = selling_price - bonus_dep;
174 e 0886      put file (output) edit('D O U B L E   D E C L I N I N G')
175 e 08A6          (page,column(indent),x(10),a);
176 e 08A6      call header();
177 e 08A9      depreciation_value = adj_price - residual_value;
178 e 08C4      book_value = adj_price;
179 e 08D4      total_depreciation = 0;
180 e 08E4          do current_year = 1 to years
181 e 0931              while (depreciation_value > 0);
182 e 0931          year_value =
183 e 0971              decimal(book_value/years,8,2) * factor+.005;
184 e 0971          if current_year = 1 then
185 e 0987              do;
186 e 0987              year_value =
187 e 09B7                  year_value * months_remaining / 12;
188 e 09B7              FYD = year_value;
189 e 09C7              end;
190 e 09C7          if year_value > depreciation_value then
191 e 09DD              year_value = depreciation_value;
192 e 09ED          depreciation_value = depreciation_value - year_value;
193 e 0A08          total_depreciation = total_depreciation + year_value;
194 e 0A23          book_value = adj_price - total_depreciation;
195 e 0A3E          call print_line();
196 e 0A5B          end;
197 e 0A5B      call summary();
198 c 0A5F      end double_declining;
199 c 0A5F
200 c 0A5F header:
201 c 0A5F      procedure;
202 c 0A5F      /* print header record */
203 e 0A5F      dcl
204 e 0A5F          new_or_used char(5);
205 e 0A5F      if new = 'yes' then
206 e 0A6F          new_or_used = ' New';
207 e 0A7F      else
208 e 0A7F          new_or_used = ' Used';
209 e 0A8B      put file (output) edit(
210 e 0B5D          '-------------------------------------------------',
211 e 0B5D              '|',selling_price+sales_tax,new_or_used,
212 e 0B5D              residual_value,' Residual Value|',
213 e 0B5D          '|',months_remaining,' Months Left ',
214 e 0B5D              tax_rate,'% Tax',tax_bracket,'% Tax Bracket|')
215 e 0B5D          (2(skip,column(indent),a),
216 e 0B5D          2(p'B$$,$$$,$$9.V99',a),
217 e 0B5D          skip,column(indent),a,x(5),f(2),a,2(x(2),p'B99',a));
218 e 0B5D
219 e 0B5D          put file (output) edit(
220 e 0B9F          '-------------------------------------------------',
```

Figure 12.4. Depreciation Schedule Program Part D.

```
221 e 0B9F      '| Y | Depreciation | Depreciation |  Book Value  |',
222 e 0B9F      '| r |   For Year   |  Remaining   |              |',
223 e 0B9F      '-------------------------------------------------')
224 e 0B9F      (skip,column(indent),a);
225 c 0B9F      end header;
226 c 0B9F
227 c 0B9F  print_line:
228 c 0B9F      procedure;
229 c 0B9F      /* print current line */
230 e 0B9F      put file (output) edit(
231 e 0C34      '|',current_year,
232 e 0C34      ' |',year_value,
233 e 0C34      ' |',depreciation_value,
234 e 0C34      ' |',book_value,'  |')
235 e 0C34      (skip,column(indent),
236 e 0C34      a,f(2),4(a,p'$z,zzz,zz9v.99'));
237 c 0C34      end print_line;
238 c 0C34
239 c 0C34  summary:
240 c 0C34      procedure;
241 e 0C34      declare
242 e 0C34          adj_ITC decimal(8,2),
243 e 0C34          total decimal(8,2),
244 e 0C34          direct decimal(8,2);
245 e 0C34      call line();
246 e 0C37      adj_ITC = ITC * 100 / tax_bracket;
247 e 0C67      total = FYD + sales_tax + adj_ITC + bonus_dep;
248 e 0C98      direct = total * tax_bracket / 100;
249 e 0CC8      put file (output) edit(
250 e 0DEE      '|      First Year Reduction in Taxable Income    |',
251 e 0DEE      '-------------------------------------------------',
252 e 0DEE      '|      Depreciation                ',FYD,'  |',
253 e 0DEE      '|      Sales Tax                   ',sales_tax,'  |',
254 e 0DEE      '|      ITC (Adjusted)              ',adj_ITC,'  |',
255 e 0DEE      '|      Bonus Depreciation          ',bonus_dep,'  |',
256 e 0DEE      '|                                  -------------  |',
257 e 0DEE      '|      Total for First Year        ',total,'  |',
258 e 0DEE      '|      Direct Reduction in Tax     ',direct,'  |')
259 e 0DEE      (2(skip,column(indent),a),
260 e 0DEE          2(4(skip,column(indent),a,
261 e 0DEE          p'$z,zzz,zz9v.99',x(3),a),
262 e 0DEE              skip,column(indent),a));
263 e 0DEE      call line();
264 c 0DF2      end summary;
265 c 0DF2
266 c 0DF2  line:
267 c 0DF2      procedure;
268 c 0DF2      /* print line of "-" */
269 e 0DF2      put file (output) edit(
270 e 0E13      '-------------------------------------------------')
271 e 0E13      (skip,column(indent),a);
272 c 0E13      end line;
273 a 0E13  end depreciate;
```

Figure 12.4. Depreciation Schedule Program Part E.

Depreciation Schedule

```
Selling  Price? 200000
Residual Value? 40000
Sales Tax (%)?  6
Tax Bracket(%)? 50
ProRate Months? 10
How Many Years? 7
New? (yes/no)   no
Schedule:
Straight    (s)
Sum-of-Yrs  (y)
Double Dec  (d)? d
List? (yes/no)  no
```

D O U B L E D E C L I N I N G

$212,000.00 Used	$40,000.00 Residual Value
10 Months Left	06% Tax	50% Tax Bracket

Yr	Depreciation For Year	Depreciation Remaining	Book Value
1	$ 35,357.14	$ 122,642.86	$ 162,642.86
2	$ 34,852.04	$ 87,790.82	$ 127,790.82
3	$ 27,383.75	$ 60,407.07	$ 100,407.07
4	$ 21,515.79	$ 38,891.28	$ 78,891.28
5	$ 16,905.27	$ 21,986.01	$ 61,986.01
6	$ 13,282.71	$ 8,703.30	$ 48,703.30
7	$ 8,703.30	$ 0.00	$ 40,000.00

First Year Reduction in Taxable Income

Depreciation	$ 35,357.14
Sales Tax	$ 12,000.00
ITC (Adjusted)	$ 20,000.00
Bonus Depreciation	$ 2,000.00
Total for First Year	$ 69,357.14
Direct Reduction in Tax	$ 34,678.57

Type RETURN to Continue

Figure 12.4. Depreciation Schedule Program Part F.

Depreciation Schedule

```
Selling  Price? ,
Residual Value? ,
Sales Tax (%)?  ,
Tax Bracket(%)? ,
ProRate Months? 8
How Many Years? ,
New? (yes/no)   yes
Schedule:
Straight   (s)
Sum-of-Yrs (y)
Double Dec (d)? y
List? (yes/no)  no
```

S U M O F T H E Y E A R S

$212,000.00 New		$40,000.00 Residual Value	
8 Months Left		06% Tax	50% Tax Bracket

Yr	Depreciation For Year	Depreciation Remaining	Book Value
1	$ 26,333.33	$ 131,666.67	$ 171,666.67
2	$ 28,214.29	$ 103,452.38	$ 143,452.38
3	$ 18,473.64	$ 84,978.74	$ 124,978.74
4	$ 12,139.82	$ 72,838.92	$ 112,838.92
5	$ 7,804.17	$ 65,034.75	$ 105,034.75
6	$ 4,645.34	$ 60,389.41	$ 100,389.41
7	$ 2,156.76	$ 58,232.65	$ 98,232.65

First Year Reduction in Taxable Income

Depreciation	$ 26,333.33
Sales Tax	$ 12,000.00
ITC (Adjusted)	$ 40,000.00
Bonus Depreciation	$ 2,000.00
Total for First Year	$ 80,333.33
Direct Reduction in Tax	$ 40,166.66

Type RETURN to Continue

Figure 12.4. Depreciation Schedule Program Part G.

Depreciation Schedule

```
Selling  Price? 310000
Residual Value? 30000
Sales Tax (%)?  ,
Tax Bracket(%)? ,
ProRate Months? 12
How Many Years? 5
New? (yes/no)   yes
Schedule:
Straight    (s)
Sum-of-Yrs  (y)
Double Dec  (d)? d
List? (yes/no)  no
```

D O U B L E D E C L I N I N G

$328,600.00 New 12 Months Left		$30,000.00 Residual Value 06% Tax	50% Tax Bracket
Yr	Depreciation For Year	Depreciation Remaining	Book Value
1	$ 123,200.00	$ 154,800.00	$ 184,800.00
2	$ 73,920.00	$ 80,880.00	$ 110,880.00
3	$ 44,352.00	$ 36,528.00	$ 66,528.00
4	$ 26,611.20	$ 9,916.80	$ 39,916.80
5	$ 9,916.80	$ 0.00	$ 30,000.00

First Year Reduction in Taxable Income

Depreciation	$ 123,200.00
Sales Tax	$ 18,600.00
ITC (Adjusted)	$ 62,000.00
Bonus Depreciation	$ 2,000.00
Total for First Year	$ 205,800.00
Direct Reduction in Tax	$ 102,900.00

Type RETURN to Continue

Figure 12.4. Depreciation Schedule Program Part H.

Depreciation Schedule

```
Selling  Price? ,
Residual Value? ,
Sales Tax (%)?  ,
Tax Bracket(%)? ,
ProRate Months? ,
How Many Years? ,
New? (yes/no)   ,
Schedule:
Straight    (s)
Sum-of-Yrs  (y)
Double Dec  (d)? s
List? (yes/no)  ,
```

S T R A I G H T L I N E

$328,600.00 New 12 Months Left		$30,000.00 Residual Value 06% Tax 50% Tax Bracket	
Yr	Depreciation For Year	Depreciation Remaining	Book Value
1	$ 55,600.00	$ 222,400.00	$ 252,400.00
2	$ 44,480.00	$ 177,920.00	$ 207,920.00
3	$ 35,584.00	$ 142,336.00	$ 172,336.00
4	$ 28,467.20	$ 113,868.80	$ 143,868.80
5	$ 22,773.76	$ 91,095.04	$ 121,095.04

First Year Reduction in Taxable Income	
Depreciation	$ 55,600.00
Sales Tax	$ 18,600.00
ITC (Adjusted)	$ 62,000.00
Bonus Depreciation	$ 2,000.00
Total for First Year	$ 138,200.00
Direct Reduction in Tax	$ 69,100.00

Type RETURN to Continue^C

Figure 12.4. Depreciation Schedule Program Part I.