7.7 Case Study: Calculating Depreciation

PROBLEM

Depreciation is a decrease in the value over time of some asset due to wear and tear, decay, declining price, and so on. For example, suppose that a company purchases a new computer system for \$200,000 that will serve its needs for 5 years. After that time, called the *useful life* of the computer, it can be sold at an estimated price of \$50,000, which is the computer's salvage value. Thus, the value of the computing equipment will have depreciated \$150,000 over the 5-year period. The calculation of the value lost in each of several years is an important accounting problem, and there are several ways of calculating this quantity. We want to write one or more functions to calculate depreciation tables that display the depreciation in each year of an item's useful life.

OBJECT-CENTERED DESIGN

BEHAVIOR. Each function should receive from its caller the amount to be depreciated and the number of years. The function should then display on the screen a depreciation table.

OBJECTS. The objects in this problem are straightforward:

	Software Objects			
Problem Objects	**Type**	**Kind**	**Movement**	**Name**
The amount to be depreciated	varying	real	received	*amount*
The item's useful life (in years)	varying	integer	received	*numYears*
The annual depreciation	varying	real	—	*depreciation*

Each function will have the same specification:

Receive: *amount* and *numYears*
Output: a depreciation table

OPERATIONS AND ALGORITHMS. The operations in this problem depend on the method used to calculate the depreciation. There are several different methods, and we will consider two of them here.

One standard method is the **straight-line method,** in which the amount to be depreciated is divided evenly over the specified number of years. For example, straight-line depreciation of \$150,000 over a 5-year period gives an annual depreciation of \$150,000 / 5 = \$30,000:

Year	Depreciation
1	$30,000
2	$30,000
3	$30,000
4	$30,000
5	$30,000

With this method, the value of an asset decreases a fixed amount each year.

The operations needed to calculate straight-line depreciation are all provided by C++ operations and statements:

i. Divide a real (*amount*) by an integer (*numYears*)
ii. Output an integer (the year) and a real (the depreciation)
iii. Repeat ii a specified number (*numYears*) of times.

Organizing them in an algorithm is straightforward.

Algorithm for Straight-Line Depreciation

```
1. Calculate depreciation = amount / numYears.
2. For year ranging from 1 through numYears do the following:
       Display year and depreciation.
```

Another common method of calculating depreciation is called the **sum-of-the-years'-digits method.** To illustrate it, consider again depreciating $150,000 over a 5-year period. We first calculate the "sum of the years' digits," 1 + 2 + 3 + 4 + 5 = 15. In the first year, 5/15 of $150,000 ($50,000) is depreciated; in the second year, 4/15 of $150,000 ($40,000) is depreciated; and so on, giving the following depreciation table:

Year	Depreciation
1	$50,000
2	$40,000
3	$30,000
4	$20,000
5	$10,000

In addition to the operations for the straight-line method, the sum-of-the-years'-digits method requires:

iv. Sum the integers from 1 through some given integer (*numYears*)

For this we can use the function `sum()` in Case Study 7.7.1.

An algorithm for displaying annual depreciation values using this method is as follows:

Algorithm for Sum-of-the-Years-Digits Depreciation

```
1. Calculate sum = 1 + 2 + . . . + numYears.
2. For year ranging from 1 through numYears do the following:
     a. Calculate depreciation = (numYears – year + 1) * amount / sum
     b. Display year and depreciation.
```

CODING. The function `straightLine()` in Case Study 7.7-1 implements the algorithm for the straight-line method of depreciation and the function `sumOfYears()` implements the algorithm for the sum-of-the-years'-digits method. Because these functions are useful in a variety of problems, we would probably store them in a library with these function definitions along with that for function `sum()` in an implementation file and the prototypes

```
void straightLine(double amount, int numYears);
void sumOfYears(double amount, int numYears);
```

in the corresponding header file (e.g., `Depreciation.h`).

Case Study 7.7-1 Depreciation Functions.

```
#include <iostream>       // cout
#include <iomanip>        // setprecision(), setw(), setiosflags()
#include <cassert>        // assert()
using namespace std;

/* straightLine() displays a depreciation table for a given
 * amount over numYears years using the straight-line method.
 *
 * Receive: amount, a real
 *          numYears, an integer
 * Output:  a depreciation table
 * Uses:    format manipulators from iostream and iomanip
 ****************************************************************/

void straightLine(double amount, int numYears)
{
   double depreciation = amount / numYears;

   cout << "\nYear -  Depreciation"
        << "\n--------------------\n";

   cout << fixed << showpoint << right       // set up format for $$
        << setprecision(2);

   for (int year = 1; year <= numYears; year++)
      cout << setw(3) << year
           << setw(13) << depreciation << endl;
}
```

```
/* sumOfYears() displays a depreciation table for a given
 * amount over numYears years using the sum-of-the-years'-digits
 * method.
 *
 * Receive: amount, a real
 *          numYears, an integer
 * Output:  a depreciation table
 * Uses:    function Sum() from Figure 6.1
 *          format manipulators from iostream and iomanip
 ****************************************************************/

void sumOfYears(double amount, int numYears)
{
   cout << "\nYear -  Depreciation"
        << "\n--------------------\n";

   double yearSum = sum(numYears);

   double depreciation;

   cout << fixed << showpoint << right       // set up format for $$
        << setprecision(2);

   for (int year = 1; year <= numYears; year++)
   {
      depreciation = (numYears - year + 1) * amount / yearSum;
      cout << setw(3) << year
           << setw(13) << depreciation << endl;
   }
}
```

TESTING. The program in Case Study 7.7-2 uses these depreciation functions. It begins by defining a menu

```
Enter:
   a - to enter information about a new item
   b - to use the straight-line method
   c - to use the sum-of-the-years'-digits method
   d - to quit
-->
```

which it passes to the function `getMenuChoice()` described in Section 7.2 of the text. This function repeatedly displays a menu and reads the user's choice until the user enters a valid choice, which is then returned to the caller. For option a, the user enters an item's purchase price, its salvage value, and its useful life. Options `b` and `c` call the functions `straightLine()` and `sumOfYears()`, respectively, to display the depreciation tables. The sentinel value `QUIT = 'd'` signals the end of input and a `switch` statement is used to process non-`QUIT` options.

Case Study 7.7-2 Depreciation Functions.

```
/*  depreciationTables.cpp computes depreciation tables.
 *
 *  Input:  purchase price, salvage value, and useful
 *          life of an item
 *  Output: depreciation tables.
 *****************************************************************/

#include "Depreciation.h"
#include <iostream>         // <<, >>, cout, cin
#include <string>           // string
using namespace std;

// or if not using this library,
// insert the prototypes of straightLine() and sumOfYears() here and
// insert after main(), the function sum() from Figure 7.2 of the text
// and the #include directives and function definitions from
// Case Study 7.7-1.

char getMenuChoice(string MENU, char firstChoice, char lastChoice);

int main()
{
   const string MENU =
        "\nEnter:\n"
        "   a - to enter information for a new item\n"
        "   b - to use the straight-line method\n"
        "   c - to use the sum-of-the-years'-digits method\n"
        "   d - to quit\n"
        "--> ";
   const char QUIT = 'd';

   cout << "This program computes depreciation tables using\n"
        << "various methods of depreciation.\n";

   char option;                          // menu option selected by user
   double purchasePrice,               // item's purchase price,
          salvageValue,                //   salvage value, and
          amount;                      //   amount to depreciate, and
   int usefulLife;                     //   useful life in years

   for (;;)
   {
      option = getMenuChoice(MENU, 'a', QUIT);  // get user's choice

      if (option == QUIT) break;
      switch (option)                 // perform the option selected
      {
        case 'a':                     // get new item information
           cout << "What is the item's:\n"
                   "   purchase price? ";
           cin >> purchasePrice;
           cout << "   salvage value? ";
           cin >> salvageValue;
           cout << "   useful life? ";
           cin >> usefulLife;
           amount = purchasePrice - salvageValue;
           break;
        case 'b':                     // straight-line method
           straightLine(amount, usefulLife);
           break;
        case 'c':                     // sum-of-years-digits method
           sumOfYears(amount, usefulLife);
           break;
        default:                      // execution shouldn’t get here
           cerr << "*** Invalid menu choice: " << option << endl;
      }
   }
}

/* getMenuChoice() repeatedly displays a MENU of choices in the
 * range firstChoice to lastChoice and reads a user's choice until
 * a valid choice is entered, which is then returned to the caller.
 *
 * Receive: MENU, a string
 *          firstChoice and lastChoice, chars
 * Return:  the choice entered by the user
 *****************************************************************/

char getMenuChoice(string MENU, char firstChoice, char lastChoice)
{
   char choice;                       // what the user enters

   for (;;)
   {
      cout << MENU;
      cin >> choice;

      if ((choice >= firstChoice) && (choice <= lastChoice))
         return choice;

      cerr << "\nI'm sorry, but " << choice
           << " is not a valid menu choice.\n";
   }
}
```

Sample run:

```
This program computes depreciation tables using
various methods of depreciation.

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> a

What is the item's
   purchase price? 2000.00
   salvage value? 500.00
   useful life? 5

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> b

Year -  Depreciation
--------------------
  1       300.00
  2       300.00
  3       300.00
  4       300.00
  5       300.00

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> c

Year -  Depreciation
--------------------
  1       500.00
  2       400.00
  3       300.00
  4       200.00
  5       100.00

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> x
```

```
I'm sorry, but x is not a valid menu choice

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> a

What is the item's
   purchase price? 1200.00
   salvage value? 200.00
   useful life? 3

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> b

Year -  Depreciation
--------------------
  1       333.33
  2       333.33
  3       333.33

Enter:
   a - to enter information for a new item
   b - to use the straight-line method
   c - to use the sum-of-years'-digits method
   d - to quit
--> d
```

Exercise

1. A third method of calculating depreciation is the *double-declining balance method*. In this method, if an amount is to be depreciated over *n* years, 2 / *n* times the undepreciated balance is depreciated annually. For example, in the depreciation of $150,000 over a 5-year period using the double-declining balance method, 2/5 of $150,000 ($60,000) would be depreciated the first year, leaving an undepreciated balance of $90,000. In the second year, 2/5 of $90,000 ($36,000) would be depreciated, leaving an undepreciated balance of $54,000. Since only a fraction of the remaining balance is depreciated each year, the entire amount will never be depreciated. Consequently, it is permissible to switch to the straight-line method at any time. Develop an algorithm for this third method of calculating deprecia-

tion.

Modify the program in Case Study 7.7-1 so that it includes this third method of calculating depreciation as one of the options. Also, modify the output produced by the depreciation functions so that the year numbers in all the depreciation tables begin with the current year rather than with year number 1.