Loan Payment Calculation Lab

From: Ima Yur Boss
To: Loyal Employee

Completion Deadline: **September 21, 2024**. Expect addition labs will be assigned prior to this deadline. Do not procrastinate! All submissions must include a cover page with at least:

a) `your complete name,
b) Descriptive title for the lab, example: Lab 1 Loan Interest,
c) The date the lab is submitted.
d) Submitted in an approved envelope.

The lab will be returned but not graded if this information is not complete!

You must use the Ada language! No alternatives for this lab! You may use Microsoft Windows, Apple or any version of UNIX/Linux.

Each individual is expected to complete all assignments during the semester individually without utilizing work from other students, code repositories, AI or other means. The penalty for not comply with this rule is typically "course failure!" Discussing problem solving approaches, helping others to master concepts, language constructs, help with debugging and related activities is encouraged!

"C" Option (maximum grade75):

Current Texas "Truth in Lending Laws" require lenders to produce a loan payment schedule for their customers. Consider the following example: Assume our customer wishes to purchase a new vehicle for $43,000 with a downpayment of $23,000. We have agreed to finance the remaining $20,000 at our standard interest rate is 6.0% for 5 years. We may offer a rate of 5.867% if they extend the loan period to 15 years. Payment schedules should help clients with the decision process determining which alternative is best for them in terms of monthly payment and total interest paid over the period of the loan.

Please write a program PDQ (pretty darn quick) to implement this government requirement. If unable to complete the assignment, please update your resume. You will be accorded the opportunity to travel and meet new people at your own expense.

Calculate the monthly interest payment as described below (the sample calculations close to Ada, Java, and C++).

A) **The "Truth In Lending Act" law requires the following customer report (on the screen) with appropriate headings to take the following form with an entry for each**

month (alternately year) of the loan. If necessary, adjust the final payment to match the exact value of the loan! The table should appear as follows for the first 12 months. Note the amount of the payment applied to the loan goes up each month till the loan is completely paid while the amount to interest goes down.

Month	Balance	Payment	Interest	Towards Loan	New Balance
1	$20,000	$387	$287	$100	$19,713
2	$19713	$387	$288	$99	$19,425
3	$19,425	$387	$290	$97	$19,136
ooo	ooo	$387	ooo	ooo	ooo
12	$16,767	$387	$303	$84	$16,464
ooo	ooo	$387	ooo	ooo	ooo
60	0	$387 - maybe	ooo	ooo	0

B) **The total interest paid is the sum of the "Interest" column and should be displayed for the customer or product of the monthly payment and number of payments!**

C) **The amount of the monthly payment (rounded to the nearest dollar for convenience and legal requirements) is $387 for this example. The formula for the calculation payment is explained below.**

D) Calculate the amount of the $387 payment applied to principal and interest as follows each payment period :

1) Divide the interest rate by the number of payments made each year, 0.06/12 = 0.005 in the example.
2) Next multiply the balance remaining from the previous month by 0.005 (in this example). Hence the amount of the payment going to interest the first month is $20,000 * 0.005 = $287.
3) Only $100 goes to pay off the principal the first month. The second month is $19713 * 0.00 5 = $288 interest an $99 to principal.

Submit a hard copy of the Truth in Lending report for use by our credit staff working with the customer.

E) **You must use I/O redirection to generate the report in a file. Include the command line used for the I/O redirection**! Remember to include the Cover Sheet, the printed copy of the Truth in Lending report created **using I/O redirection** followed by a copy of your code. **All submissions must be in envelopes large enough to hold the lab without folding.** Do not glue the envelope closed or fasten any clasp. Be sure to include your name, class with section number and day class meets, and date submitted.

Next:

Use your lab to determine interest on a house loan for $250,000 at 7.75% interest for 20 years and 30 years. **Submit only the total interest for each loan clearly labeled!** You should see a substantial difference in the amount of interest paid.

"B" Option (maximum grade85):

Many customers would like a hard copy of the loan report to take home. We anticipate the Texas Legislature to make this a legal requirement in the next legislative session. We will be obligated to retain a copy of the "Truth in Lending" report as part of the file kept on each active loan for 5 years after the loan is paid off. We would naturally prefer an electronic copy of the report as opposed to a paper copy.

You must implement the actual loan calculations as a procedure like "<procedure-name>(principal, rate, years, interest)." The main program should prompt and obtain the loan values from the customer the call the procedure to generate to do the calculations and produce the report. The total interest must be returned to the main program for display/printing by "reference" (inout). All other values must be passed by "value" (in). (This corresponds to a "C"/C++/Java function with no return statement.

You must modify the "C" Option such that it not only prints the report on the screen but also writes it to a sequential text file. We can print as many copies of the text file as desired by the customer and retain an electronic copy for our records. It would be nice if we could specify the name of the report file at runtime. For example, "JonesCarLoanOctober2024." Do not forget to formally open/close the file when finished.

Submit a copy of the report meeting the "C/B" option submission requirements.

"A" Option (maximum grade 100):

Complete the "C" and "B" options but use an Ada function to calculate the "monthly loan payment." The form of the functions will be "function <name of function(parameters passed by value) returns Float" or Long_Float (float and double in C++/Java) with two digits to the right of the decimal point. Ada enforces the abstraction for functions returning a single value the parameters must be of type "value/in" in the parameter list. C++ and Java have the same specification in the language their ISO standard but most compilers/interpreters do not enforce it. They assume programmers are intelligent, knowledgeable, trainable and will adhere to the standard. The reason for the standard is to ensure call parameters will not be modified in the function. Therefore, we cannot be subjected to potential side effects due to function with a return type improperly changing the value of the parameters in its body. This improves program reliability and greatly reduces time in debugging. If a C++ or Java programmer needs to change the value of parameters in a function, parameters to be changed should be passed by reference and the function should not have a return statement. Parameters that should not be changed should be passed by value. **Your submission must meet all "C/B" option requirements.**

As a matter of interest Ada provides more power for passing parameters than C++ and Java including "in," "out, "inout" and the ability to pass arrays and other structured types by value or reference. If an array is passed by value, it does not require additional storage or a copy operation from the main program to the function when the Ada function is invoked. The compiler enforces the "CBV" abstraction by not allowing code to assign values to array elements (appearing on the left of the assignment operator ":="). This is particularly useful in real time programming. For example, the main program for a spacecraft pass an array of information to separate independently executing navigation module/task. If the navigation task fails, the main program can restart the navigation module with the knowledge no require data has been accidentally modified.

Method to Calculate Monthly payment using Java, C++ or Ada

The theoretical value for the monthly payment is:

Monthly Payment =

(monthlyInterestRate * Principal * (1.0 + monthlyInterestRate)**N /

(1.0 + monthlyInterestRate)**N – 1)

Or more closely to code in Java or C++:

Monthly Payment = (0.06/12* principal * Math.pow(1+ 0.06/12 , 60)) /

(Math.pow(1 + 0.06/12, 60) - 1) = $386.656 or $386.66 to the nearest cent or $387 to the nearest dollar. Ada uses "**" to indicate exponentiation.

Ada, C++, Java and Excel values are just approximations. I used Excel on a 64 bit computer rounding to the nearest cent. The JVM (Java Virtual Machine) is restricted to 32 bits (or as close as it can come on the physical architecture. While not obvious, the exponential portion of the above calculation for both architectures results in values exceeding <float>x10^{74}. The exponential portion of the calculation was close to 1.0 (dividing numerator by the denominator) hence both Excell and the JVM produced the same result to the nearest cent. Fortunately, the division of the exponents approaches 1.0.

Put another way the exponential portion of the calculation for the example required 75 or more significant digits. A 32 bit (Java) architecture can only approximate 7 significant decimal digits while the 64 bit architecture can support approximately 16 decimal digits. The extra digits required in the calculation were treated as zeros. We require only payments to the nearest dollar in most financial calculations. If the value of the principal, interest rate and number of years have too many significant digits, calculations will not be accurate.

There are two solutions. The first is to write your own routines to do arithmetic on large numbers. As an example, assume the equation A = B * C where A, B, and C are arrays with 300 locations each. If you store one digit per array entry, you have the ability to represent 300 digit numbers at one digit per array element. Addition is simply A(J) = B(J) + C(J) with values greater than 10 resulting in a carry to A(J - 1) [carry to the left]. Multiplication using the arrays is simply a series of additions. Division is a series of subtractions. Sometimes programmers must find ways to exceed the limitations of the architecture we are working on (Python).

The second solution is to use Java's big integer and big decimal classes. Hint: One way to implement this ability in Java and Python appears in the preceding paragraph. The arrays are simply hidden from the programmer. Some non-real time languages like Python provide a fairly wide range without programmer knowledge or intervention.

To Verify the Value for the Loan Payment:

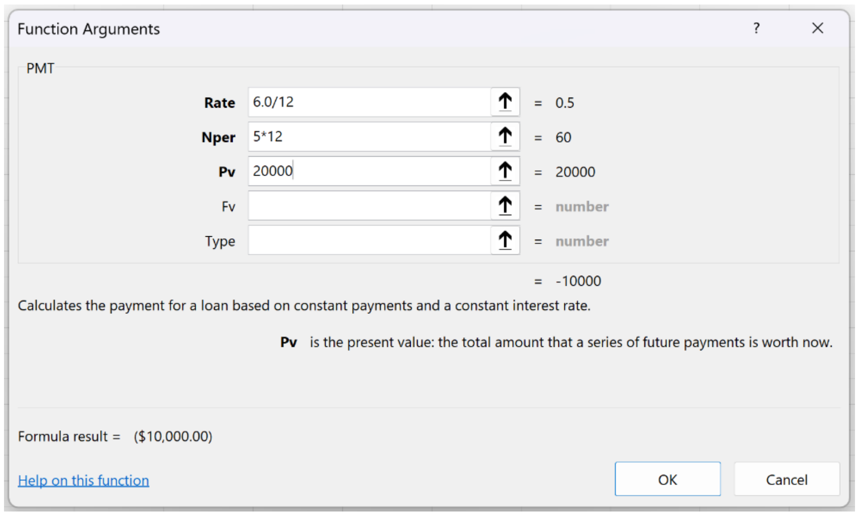
You may verify your Ada code is accurate for the loan payment using Excel or another spreadsheet. To obtain the monthly loan payment required to reduce the loan to zero at the end of the loan period in Microsoft Excel, use the financial function PMT as follows:

To calculate the interest rate for the loan using a Microsoft Excel spread sheet:

1) Click in any cell. Select Insert, Formula, Financial, **PMT**
2) Enter the loan values as shown below.
3) When you click okay, the monthly payment will be ($386.66). The approximate value of the monthly payment is in red to indicate it is the monthly payment. Payments are typically rounded up to the nearest dollar, $387 as in this example.
4) If necessary, the last monthly payment is typically adjusted to make the total value paid identical to the initial loan value.

The computation of the monthly payment is complicated by the fact that the amount that goes to interest and to principal over the course of the loan is time dependent. For convenience, we will use MS Excel to accomplish the task. A sample calculation for the above loan follows:

After completing step one of the above Excel instructions "Click in any cell "Select Insert, Formula, Financial, **PMT"** the following will be displayed in the spreadsheet. Complete the remaining steps.



Function Arguments

PMT

Rate	6.0/12	=	0.5
Nper	5*12	=	60
Pv	20000	=	20000
Fv		=	number
Type		=	number

= -10000

Calculates the payment for a loan based on constant payments and a constant interest rate.

Pv is the present value: the total amount that a series of future payments is worth now.

Formula result = ($10,000.00)