Program 1: Simple Mortgage Calculator...

Due date: Monday, January 30, 11:59pm.

1 Purpose

To write a program requiring use of simple functions, variable declarations, assignment statements, conditional statements and standard C library output functions.

Programming environment. This is a solo programming project. You are responsible for all the work related to the program development, testing and submission.

Collaboration. **Any** collaboration between peers, as well as any collaboration with outside sources is **strictly prohibited**. If you have any questions, concerning the assignment, please consult the instructor.

2 Program Description

You are writing a simple mortgage calculator. The user of your program is planning to purchase a house and wants to find out how much the payment would be. The user supplies the program with the terms of the mortgage, and your program computes the overall amount of money the user will owe the bank over time, the monthly payment, the annual payment, and a few other things.

A formal description of the program is given below.

Program Requirements.

TR0. Economics. Recall from high school how the mortgage-like financial instruments operate.

Parameter name	Notation	Explanation
House price	h	price of the house
Principal	p	amount of money borrowed
Interest Rate	r	% of principal to be repaid each year
Term	t	number of years for the loan

Using this information, the monthly payment on the loan is calculated as follows. First, we compute the monthly rate. Assuming $0 \leq r \leq 100$,

$$mr = \frac{r}{100 * 12}.$$

The total number of payments on the loan is:

$$N = 12 * t.$$

Now, the monthly payment is:

$$mPayment = \frac{p \cdot mr}{1 - (1 + mr)^{-N}}.$$

The annual payment is:

$$aPayment = mPayment \cdot 12.$$

The rest of the computations you will perform in the program are going to be straightforward.

The mortgage calculator will only compute the terms of a standard fixed-interest rate loan.

TR00. Files. Traditionally, software written in C is organized into multiple files. In prior labs you have already seen how code can be put into `.h` header files, which can then be `#include`d into a C program. For this program, you will go one step further, and actually *do it right.*

In general, *header files* (i.e., files with the `.h` extension) are reserved for *declarations only.* They can be used for code, but this is not commonly practiced (We used and may continue using `.h` for code in our labs to simplify your development and to simplify grading).

You will write this program in three different files.

1. `mortgage.c`: this file will contain all functions that constitute the mortgage calculator. This file, however, **shall contain NO** `int main()` function.
2. `testMortgage.c`: this file will contain a single `int main()` function, which will, in turn, contain a list of tests you have used to test your mortgage calculator.
3. `mortgage.h`: a header file that will contain all preprocessor instructions (e.g., `#define`d constants) and all function declarations for all functions defined in `mortgage.c`.

Additionally, you will submit a fourth file:

- `unitTests.c`. This file shall contain `checkit_double()` unit tests of all functions from `mortgage.c` that return values.

More on specifics of each file below.

When a C program consists of multiple files, it is "assembled" as follows.

- All header files are `#include`d in all `.c` files. Your `mortgage.c` and `testMortgage.c` files each shall contain the

  ```
  #include "mortgage.h"
  ```

  This "informs" the compiler about the functions that are available for use in the program. In your program, the functions are declared in the `mortgage.h` file and are defined in the `mortgage.c` file. (see below for the specifications for the functions you need to define).

- The program is compiled using the following command:

  ```
  > gcc -ansi -Wall -Werror -lm -o <name> <file1.c> <file2.c> ... <fileN.c>
  ```

  In the case of your program, you will compile it using the following command:

  ```
  > gcc -ansi -Wall -Werror -lm -o testMortgage testMortgage.c mortgage.c
  ```

  The C compiler will do the rest.

TR1. Overview. The code for the mortgage computations will reside in the `mortgage.c` file. This file will contain function definitions for the following functions:

Function	Note
`void mortgage(...)`	takes as input mortgage parameters, prints report
`double getDownPayment(...)`	computes down payment
`double getDownShare(...)`	computes share of the down payment
`double getMonthlyPayment(...)`	computes monthly mortgage payment
`double getTotalBankPayment(...)`	computes lifetime mortgage payment

These functions are described in detail below. Note, that it is **your job** to develop correct function declarations for each function based on the information provided to you in this document.

TR2. `void mortgage()`. The `void mortgage()` function takes as input four parameters describing the terms of the mortgage:

1. House price
2. Loan amount
3. Interest rate
4. Loan term

While the exact declaration of the function is left up to you, the order of input parameters in the function declaration must match the order above (i.e., house price is #1, loan amount is #2 and so on).

This function works in three steps.

On **Step 1** it tests all input parameters for correctness. If the value of any parameter is determined to be incorrect, `void mortgage()` shall print an error message and stop further execution.

On **Step 2** the function performs computations to determine the following information:

1. Monthly payment;
2. Annual payment;
3. Total amount to be repayed;
4. Total interest payments;
5. Interest as % of principal;
6. Down payment;
7. Total amount spent on the house over time;

Some of these computations shall be performed by calling the appropriate functions defined in `mortgage.c` file. The remaining computations shall be performed directly in the body of the `void mortgage()` function.

On **Step 3** the function outputs a mortgage report, printing all input and computed information about the mortgage.

TR3. Input parameters. `void mortgage()` takes as input four parameters. The types and units of measurement for them are specified in the table below.

Parameter	Type	Unit of measurement	Example
House price	`double`	*Dollars*	5000000.00
Loan amount	`double`	*Dollars*	4000000.00
Interest reate	`double`	*per cent*	5.5
Loan term	`int`	*Years*	30

`void mortgage()` and some other functions will use local variables to contain results of various computations. All these variables shall have `double` data type.

TR4. Output. `void mortgage()` shall print numerous lines using `printf()` calls throughout its body. **All output generated by this function must match the specifications exactly!** The output of the instructor's program shall be provided to you for comparison. The capitalization, spelling and spacing of all text must coincide. All floating point (double) numbers shall be truncated at two digits past the decimal point using the `%.2f` format template.

TR5. Step 1. Parameter checks. Prior to engaging in computations, your `void mortgage()` function shall perform checks of the validity of values of all its input parameters. The checks shall be performed in the order in which the parameters are listed in requirement **TR2.** The following table specifies when the input parameters are valid[1]:

[1] These conditions are used for the purpose of this assignment. They are not always true. E.g., a 0% down mortgage can potentially finance an amount that is larger than the price of the house.

Parameter	Validity condition(s)
House price	*non-negative*
Loan amount (principal)	*non-negative, less or equal to house price*
Interest rate	*between 0 and 100 (inclusively)*
Term	*positive*

Your function shall print the following text, if an input parameter fails the appropriate check:

1. House price is negative. `void mortgage()` shall print

   ```
   Full price: Incorrect value
   ```

2. Loan amount exceeds house price. `void mortgage()` shall print

   ```
   Principal greater than full price
   ```

3. Loan amount is negative. `void mortgage()` shall print

   ```
   Principal: Incorrect value
   ```

4. Interest rate is negative. `void mortgage()` shall print

   ```
   Interest: Incorrect value
   ```

5. Term of loan is non-positive. `void mortgage()` shall print

   ```
   Term: Incorrect value
   ```

Each printed line shall end with a newline character. `void mortgage()` shall stop its work after detecting and reporting any of the errors above.

TR6. Step 2. Calculations.

If the input parameters pass all checks, `void mortgage()` shall perform the following computations.

- The down payment (difference between the house price and the loan amount). This is performed by calling the `getDownPayment()` function and storing the result.
- The share of the downpayment to the full price of the house (in per cent out of 100). This is performed by calling the `getDownShare()` function and storing the result.
- The monthly payment on the loan. This is performed by calling the `getMonthlyPayment()` function and storing the result.
- The annual payment on the loan. This is computed directly in the `mortgage()` function.
- Total amount of money needed to repay the loan. This is performed by calling the `getTotalBankPayment()` function and storing the result.
- Total interest payment (total amount of money to repay the loan, minus the amount of the loan principal). This is computed directly in the `mortgage()` function.

- Total amount of money that will spent on the house (including the down payment). This is computed directly in the `mortgage()` function.
- The ratio of the total amount of interest payments to the loan principal as a per cent out of 100. This is computed directly in the `mortgage()` function.

Use requirement **TR0.** for the math behind these computations. Any computations not specified there should be straightforward. See requirements below for the specifications of the remaining functions.

TR7. Step 3. Mortgage Calculator Report. After performing the computations, `void mortgage()` shall print its report.

First, `void mortgage()` shall skip one line, and then print:

```
-------------- Report -----------------
```

(make certain you use the correct number of dashes)

After that, your program will output the following lines:

1. The first line:

   ```
   House price: $
   ```

   followed by the price of the house (the first input parameter to the `void mortgage()` function).

2. The second line:

   ```
   Amount borrowed: $
   ```

   followed by the price of the house (the second input parameter to the `void mortgage()` function).

3. The third line:

   ```
   Interest rate: XXXX;   Term: YY years
   ```

   where `XXXX` is replaced with the interest rate value (the third input parameter to `void mortgage()` and `YY` is replaced with the term of the loan value (the fourth input parameter to `void mortgage()`).

   This is followed by an empty line.

4. The fifth line:

   ```
   Monthly payment: $
   ```

   followed by the amount of the monthly payment.

5. The sixth line:

   ```
   Annual payment: $
   ```

   followed by the amount of the annual payment.

6. The seventh line:

   ```
   Total amount to repay: $
   ```

   followed by the amount the user would have to pay the bank.

7. The eighth line:

   ```
   Total interest payments: $
   ```

   followed by total amount of the interest payments.

8. The nineth line:

   ```
   Interest is XX.XX% of the principal
   ```

   where you shall insert the actual percentage you compute in place of `XX.XX`. (note the "%" character in the output).

9. Leave the tenth line empty.

10. The eleventh line:

    ```
    Down payment: $XXXXXX is YYYY% of house price
    ```

    where `XXXXX` is replaced with the down payment amount and `YYYY` is replaced with the percentage the down payment is of the full house price.

11. The twelfth line:

    ```
    Total spent on the house: $
    ```

    followed by the total amount the user will spend on the house.

End the report with an empty line, followed by the line shown below:

```
----------------------------------------
```

Finally, place two more empty lines below the line of dashes.

All numbers should be reported as floating point numbers with two decimal places (payments are always made in terms of dollars and cents).

TR8. getDownPayment(). The `double getDownPayment()` function takes as input two parameters: the price of the house and the amount of the loan (loan principal). It returns the size of the down payment on the house.

TR9. getDownShare(). The `double getDownShare()` function takes as input two parameters: the price of the house and the size of down payment, and reports the percentage (a number from 0 to 100) the down payment is from the total house price.

TR10. getMonthlyPayment(). The `double getMonthlyPayment()` function takes as input three parameters: the amount of the loan (loan principal), the interest rate and the loan term. It returns the monthly mortgage payment.

The computation should proceed according to the formulas found in requirement **TR0**. It may involve computing a number of temporary quantities and storing them in local variables (e.g., the monthly interest rate and the total number of payments).

TR11. getTotalBankPayment(). The `double getTotalBankPayment()` function takes as input two parameters: the monthly mortgage payment, and the loan term. It produces the total amount of money that must be repaid to the bank over the lifetime of the mortgage.

TR12. Unit testing. You shall unit-test the four functions:

```
double getDownPayment();
double getDownShare();
double getMonthlyPayment();
double getTotalBankPayment();
```

For each function you shall assume that all input parameters have valid values (i.e., that any values have already been tested).

Create a file `unitTests.c`, which contains a single `int main()` function, which, in turn, contains a collection of your unit tests. A unit test in the context of this assignment is the use of a `checkit_double()` macro to test the values returned by a specific function.

For `getDownPayment()`, `getDownShare()` and `getTotalBankPayment()` you shall include at least five (5) unit tests. For `getMonthlyPayment()` you shall include at least 10 unit tests, possibly more, if you feel more tests are required to properly test the function.

You are responsible for creating all test cases. Each invocation of `checkit_double()` macro shall contain a call to one of the functions above as its first argument, and the correct value the function shall return (computed by you) as the second argument (i.e., it has to be a single number).

TR13. Program testing. Your `void mortgage()` function is tested by a `testMortgage.c` program.

The `testMortgage.c` file shall contain the appropriate preprocessor directives, and a single function, `int main()` which contains a number of calls to the `mortgage()` function is a variety of parameters.

You are provided with a version of the `testMortgage.c` file with a collection of public tests. The output of the instructor's implementation on these tests is also made available. You shall:

- Complete the `testMortgage.c` file so that it compiles properly.
- Ensure that all tests run correctly, i.e., produce the output that is not distinguishable from the instructor's output.

Please be aware that the instructor also has a private set of test cases for the `mortgage()` function. Because of this, in your testing, you should not rely solely on the public tests provided by the instructor. You should develop your own test cases. You are not required to submit them - simply submit the properly working `testMortgage.c` file containing all the instructor-provided test cases, but privately, you should test your program on a larger number of cases.

General Notes

Math. You are responsible for the remainder of the program design for this program. In particular, you are responsible for coming up with the correct math to compute the outputs of the program based on the inputs, whenever the math is underspecified. (Mostly, the computations that were not provided to you are straightforward).

ANSI C. Your program shall be written in ANSI C. The instructor will compile your program using the following `gcc` flags:

```
gcc -ansi -Wall -Werror -lm
```

(note that you may need to use `pow()` function from the `math.h` library package, hence `-lm` flag may be needed when you compile.)

Any program that does not compile in this fashion will be assigned a score of 0.

Style. Your code will checked for style. Your program shall conform to the style described at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

In addition, the header comment shall be as described in Lab 2 specification.

Any style violations are subject to an automatic 10% penalty.

Testing. As mentioned above, you are provided with the instructor's output for the public set of test cases, stored in the file `testMortgage-alex.out`

To test whether your program returns the same results, compile your program, and then run it while *redirecting the output of the program to a file*. After that, you can use Linux's `diff` command to compare the two output files. The commands are as follows:

```
> gcc -ansi -Wall -Werror -lm -o testMortgage testMortgage.c mortgage.c
> testMortgage > my.out
> diff my.out testMortgage-alex.out
```

(make sure you download `testMortgage-alex.out` to the same directory you use for your Program 1 code).

If your output coincides with the instructor's output, the `diff` command above **produces no visible output**.

Any program that fails any of the public tests will not receive more than 30% of the grade.

3 Submission Instructions

Submission.

Files to submit. You shall submit the `testMortgage.c`, `mortgage.c`, `mortgage.h` and `unitTests.c` files.

No other files shall be submitted.

Submission procedure. Use `handin` program to submit your work. The command is as follows:

```
> handin dekhtyar program01 <files>
```

(as usual, you must be logged to `unix1`, `unix2`, `unix3` or `unix4` to be able to run `handin`).

Late submission. You may submit late for a 24-hour period following the deadline. Late submissions are subject to the standard 10%—30% penalty at the instructor's discretion.