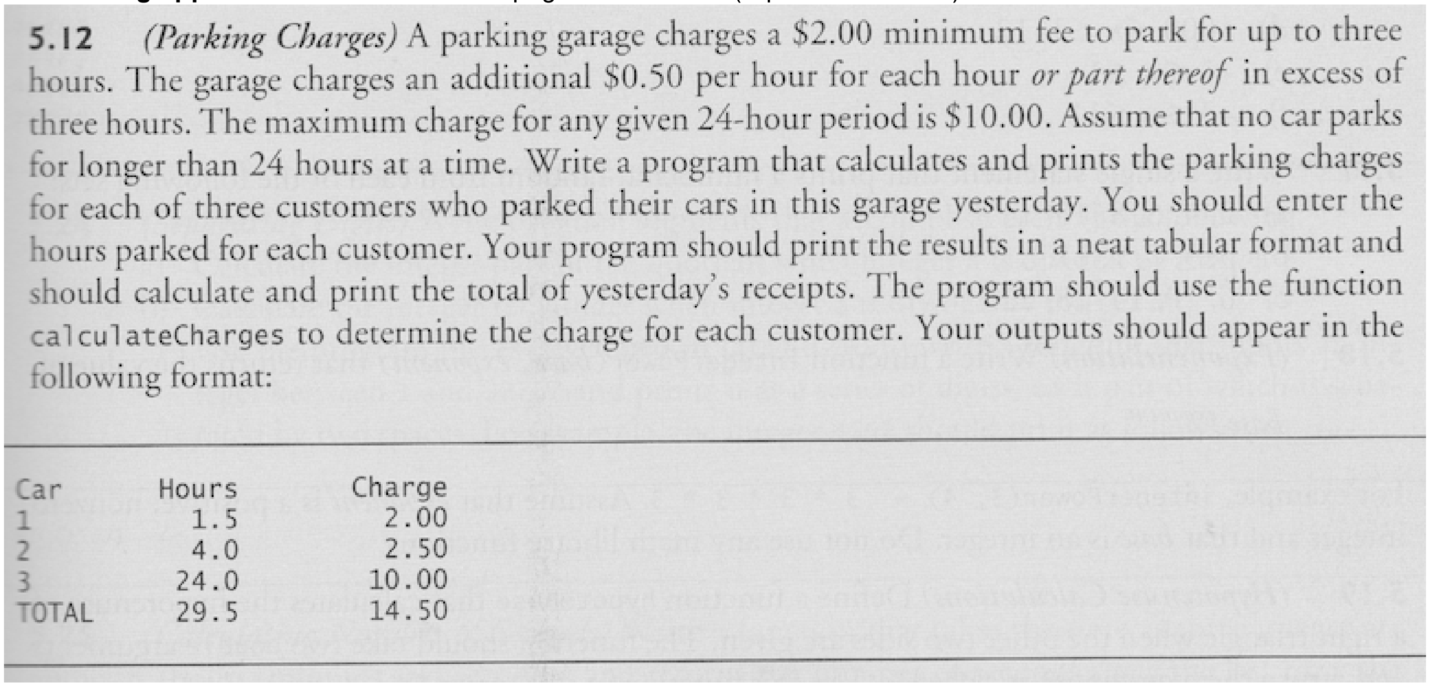
COMSC-165 Lab 5
Advanced C++ Functions

In this lab you will practice writing functions. Details of the program design are *not* explicitly given -- you need to use common sense to decide data types, prompts, output labels, etc. Refer to the part of chapter 5 where some library functions were explained, and use them where appropriate.

As you complete this assignment, post the required file(s) to the COMSC server. For assignments that involve more than one file, you may post them all at once, or one-at-a-time as you complete them, as you prefer. For assignments that involve more than one separate "lab" (like lab 1a, 1b, and 1c), the individual labs are graded in order, starting with "a", which must be fully correct before "b" is graded, and so on. This assignment is worth 50-points, to be awarded after all labs are successfully completed. Use the "Submit your FA2014 work" link on the class website to post your file for this lab, using the **lab5** folder provided.

LAB 5a: Parking Charges [`Parking.cpp`]
Write **Parking.cpp** to solve exercise 5.12 on page 223 of Deitel (reproduced below):



5.12 *(Parking Charges)* A parking garage charges a $2.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour *or part thereof* in excess of three hours. The maximum charge for any given 24-hour period is $10.00. Assume that no car parks for longer than 24 hours at a time. Write a program that calculates and prints the parking charges for each of three customers who parked their cars in this garage yesterday. You should enter the hours parked for each customer. Your program should print the results in a neat tabular format and should calculate and print the total of yesterday's receipts. The program should use the function `calculateCharges` to determine the charge for each customer. Your outputs should appear in the following format:

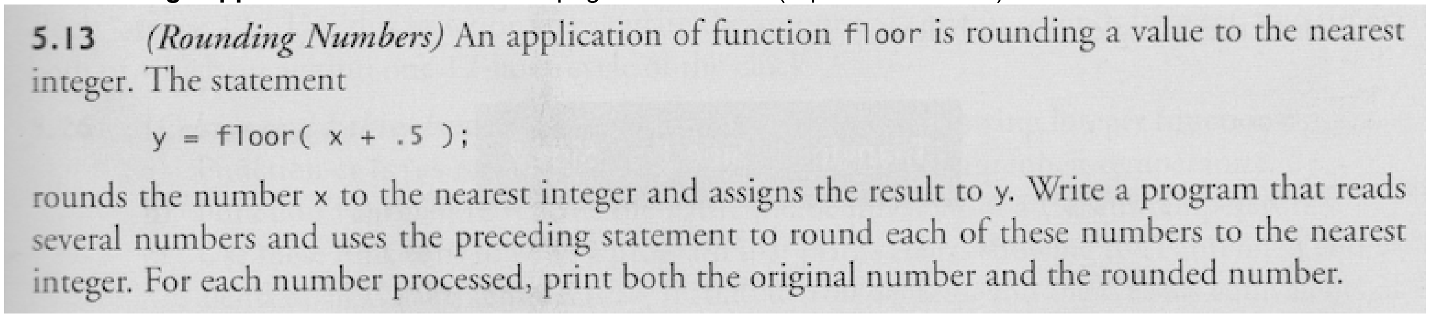
```
Car       Hours       Charge
1           1.5         2.00
2           4.0         2.50
3          24.0        10.00
TOTAL      29.5        14.50
```

"Enter the hours parked" means to use console input. "Print" means output to the console screen.

NOTE: Write your program for ONE car. No tabular output, no multiple cars, no loop. The focus should be on *functions* and I don't want other requirements to distract.

*Post **Parking.cpp** to the COMSC server for credit.*

LAB 5b: Rounding Numbers, v.1.0 [`Rounding1.cpp`]
Write **Rounding1.cpp** to solve exercise 5.13 on page 223 of Deitel (reproduced below):



5.13 *(Rounding Numbers)* An application of function `floor` is rounding a value to the nearest integer. The statement

```
y = floor( x + .5 );
```

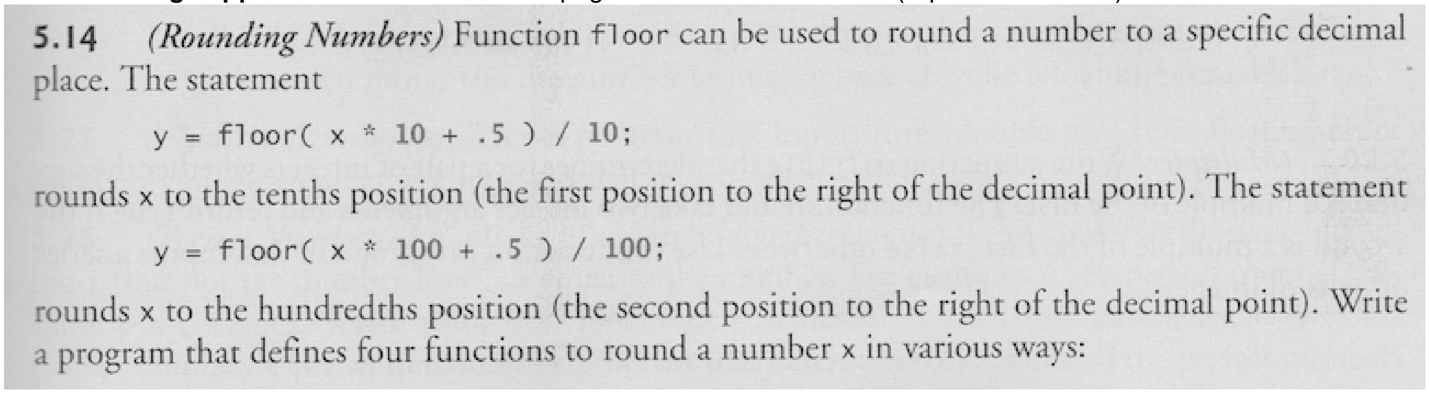
rounds the number x to the nearest integer and assigns the result to y. Write a program that reads several numbers and uses the preceding statement to round each of these numbers to the nearest integer. For each number processed, print both the original number and the rounded number.

"Reads several numbers" means to use console input, in a loop. "Print" means output to the console screen. Exit the loop when the user enters a q or Q.

Do NOT use `setprecision`, `fixed`, or `cout.setf`. Input values with more than 6 digits will be rounded (half-evenly) to 6 digits, and that's ok.

*Post **Rounding1.cpp** to the COMSC server for credit.*

LAB 5c: Rounding Numbers, v.2.0 [`Rounding2.cpp`]
Write **Rounding2.cpp** to solve exercise 5.14 on pages 223 and 224 of Deitel (reproduced below):



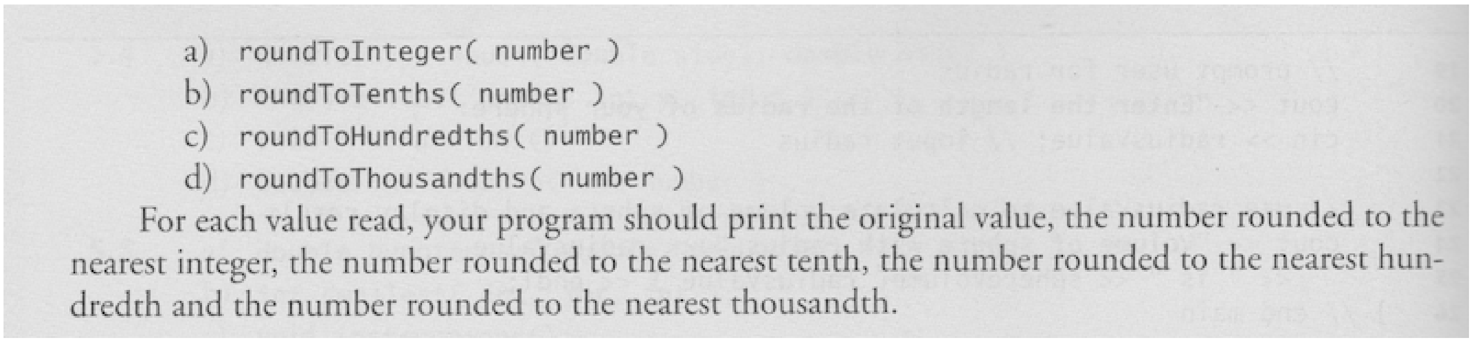
5.14 *(Rounding Numbers)* Function `floor` can be used to round a number to a specific decimal place. The statement

```
y = floor( x * 10 + .5 ) / 10;
```

rounds `x` to the tenths position (the first position to the right of the decimal point). The statement

```
y = floor( x * 100 + .5 ) / 100;
```

rounds `x` to the hundredths position (the second position to the right of the decimal point). Write a program that defines four functions to round a number `x` in various ways:



a) `roundToInteger( number )`
b) `roundToTenths( number )`
c) `roundToHundredths( number )`
d) `roundToThousandths( number )`

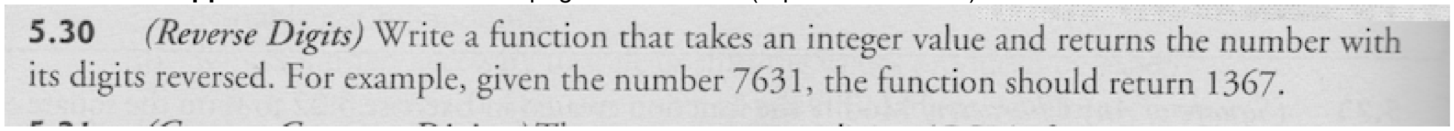
For each value read, your program should print the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth and the number rounded to the nearest thousandth.

"For each value read" means to use console input, in a loop. "Print" means output to the console screen. Exit the loop when the user enters a q or Q.

If you use `double`s, you may use `setprecision(15)`. Do NOT use `setprecision` if you use `float`s. But do NOT use `fixed` or `cout.setf`.

*Post **Rounding2.cpp** to the COMSC server for credit.*

LAB 5d: Reverse Digits [`Reverse.cpp`]
Write **Reverse.cpp** to solve exercise 5.30 on page 226 of Deitel (reproduced below):



5.30 *(Reverse Digits)* Write a function that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the function should return 1367.

Get the console input in main. Use a value-returning function to convert the input number to its reverse. Then in main, output the function's returned value, nicely formatted. Include a replay loop, and exit it when the user enters a q or Q when prompted for an input number.

Expect the input to be from 1 to 6 digits in length (inclusive). Do NOT consider leading zeros like you did in Palindrome.cpp. So the reverse of `7` is `7`. The reverse of `123` is `321`. The reverse of `123456` is `654321`.

*Post **Reverse.cpp** to the COMSC server for credit.*

GRADING POLICY
Lab work will be returned for you to fix and resubmit for any of the following reasons, or for not following stated specifications, unless directed otherwise by your instructor, Point penalties will be assessed per the syllabus for each time a

lab is returned for redo for any listed reason below or specification above. Points will be awarded for an assignment upon final, successful completion of all of its labs. Note that you will NOT receive a list of corrections to make, because to do so would remove any incentive for students to do quality work. Instead, grading of your work will stop upon discovery of the first mistake. Grading will not continue until the mistake is corrected. So to avoid multiple cycles of correction and resubmission, and the cumulative point penalties, make sure that your work is as correct as you can make it before submitting it.

Basic checklist, even if your compiler "forgives" doing otherwise:

1. Spell and case filenames *exactly* as specified. To avoid errors, be sure to configure your Windows computer to NOT hide file name extensions -- refer to the Burns Intro to Programming text, chapter 2.
2. Submit files to the specified folder. Do NOT submit archive files, like ZIP, JAR, or RAR.
3. Include identifying information as the first lines in the file AND as the first lines of your output in main.
4. Do NOT use the statement `system("PAUSE");` or any other system-specific code.
5. Do NOT try to access the value stored in an uninitialized variable.
6. CPP and H files must be free of careless typing mistakes that would prevent compilation.
7. Put an `endl` as the last item in the last-executed `cout` statement before the closing curly brace of `main`.
8. Do NOT forget to use the `cin.ignore` statements for non-string console input, unless specifically directed otherwise.
9. Do NOT forget to use the `fin.ignore` statements for non-string file input, unless specifically directed otherwise.
10. For file I/O, NEVER include a path designation. ALWAYS use the "working folder" for files.
11. Use the proper `#includes` and `usings` as listed in the C++ Library PDF.
12. Do NOT #include C-libraries such as `math.h` -- include their C++ equivalents instead, like `cmath`.
13. Do NOT #include libraries that you do not use.
14. Do NOT #include "**stdafx.h**"; do NOT use `pragma`.
15. Do NOT use C/C++ constructs or library items that are not specifically taught in this class, such as `goto`, `scanf` or `fscanf`, `fixed`, `strcmp_c`, `stricmp`, etc.
16. Do NOT use `void main()` -- use `int main()` always.
17. Do NOT use `return 0;` as the last statement in `int main()`, because it is not required by ANSI Standard C++.
18. If you use output statements in your program for debugging purposes, be sure to delete them before submitting your work.
19. Do *not* use "global variables".

Advanced checklist, even if your compiler "forgives" doing otherwise:

1. `NULL` is ONLY a pointer value -- do NOT use it for integer or other numeric values, even if your compiler allows it.
2. Do NOT declare a static array with a variable size specification, such as `int n =...; int a[n];`, even if your compiler allows it.
3. NEVER #include a CPP file, even if the textbook includes examples of such.
4. For console input and text file input with `>>`, use `.ignore` as explained in the Burns Intro to Programming text, chapter 5, unless specifically directed otherwise.
5. NEVER use `cin >>` to read directly into a numeric variable, like `int` or `double` or `float`. Read as a string (C or C++) and convert to a number using `atoi` or `atof`. Include `cstdlib`!
6. Do NOT use the `inline` keyword. To make class member functions inline, write their definitions inside the class definition.
7. Once the `const` keyword is introduced in this course, use it where applicable. Note that class "getter" functions are not really getter functions unless they have a trailing `const` keyword to flag them as such.
8. Make sure that all value-returning functions return a valid value under any logical circumstance.
9. Exit functions with `return` statements, and NOT `exit` -- do NOT use `exit` under any circumstances.
10. In every H and CPP file, ALWAYS #include libraries and H files for ALL functions, classes, and identifiers used in that file. Exception: use a class "forward declaration" instead of #including the class' H file when only pointers and reference variables appear.
11. Include the `cstring` library for char-array-based string ("C string") *functions*, and the `string` library for STL-based-strings ("C++ strings").
12. When using the random number generator seed function `srand`, call it ONCE ONLY in your program, and "sacrifice" the first value, like this: `srand(time(0)); rand();` at the top of main. Be sure to #include both `ctime` and `cstdlib`.
13. In COMSC 210, use `using namespace std;`. In 165 and 200, use `using std::` statements, properly placed under the #includes of the libraries to which they pertain.
14. `std::string` and `std::getline` belong to the `string` library. In 165 and 200, use `using std::` statements for

these. Note that `cin.getline` is a different thing, and only requires `using std::cin;`.

15. Do NOT use `using std::` for anything in the C libraries. Use it for items from C++ libraries, only.
16. Test drivers MUST at least include ifndef testing, object copy testing with assignment after declaration, and const object testing with assignment upon declaration.
17. Test drivers must NOT include console or file input, unless directed otherwise.
18. Do NOT allow memory leaks, unless specifically instructed otherwise.