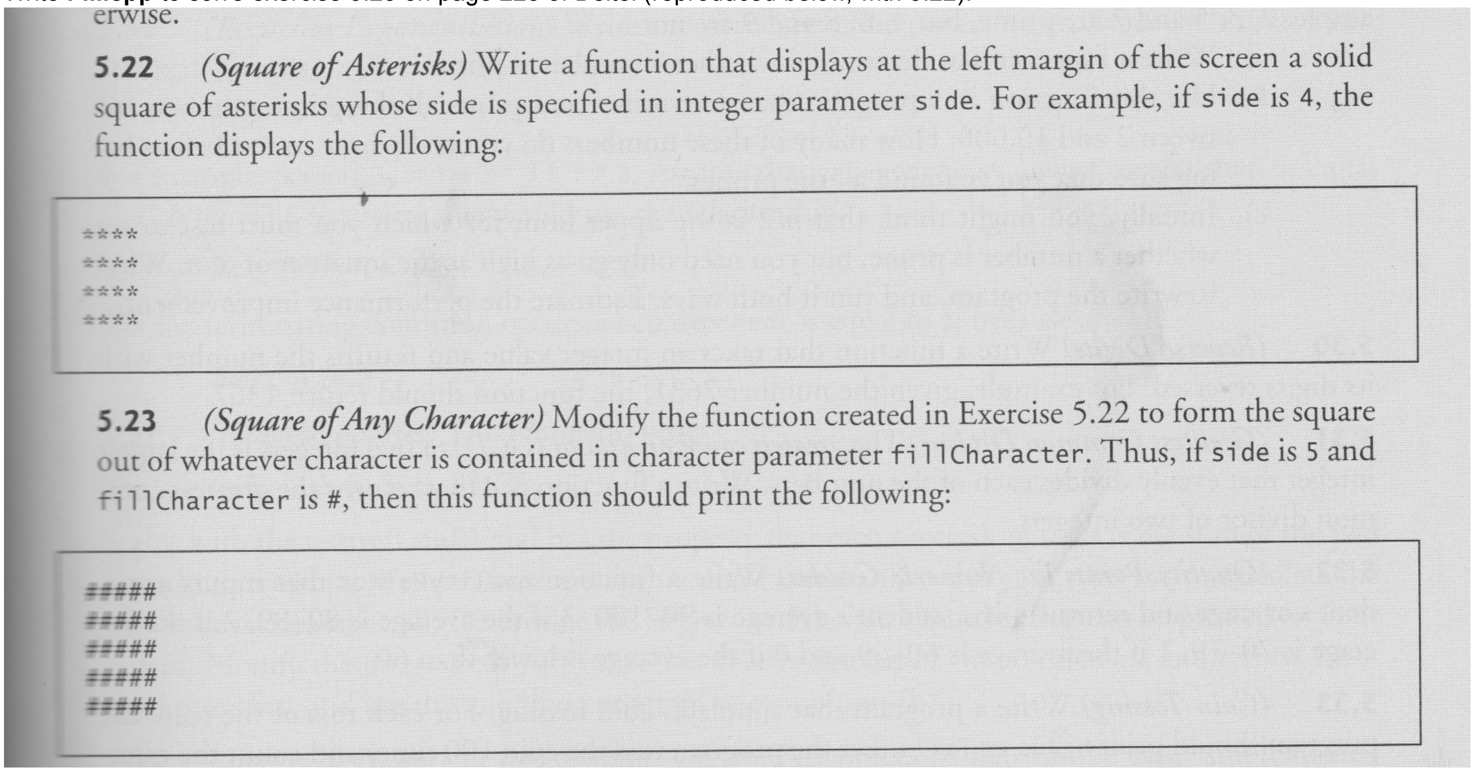
COMSC-165 Lab 4
Basics of C and C++ Functions

In this lab session, you will practice writing functions, using three exercises out of the Deitel textbook. Each exercise allows you to write a programming solution "from scratch", without a lot of guidance. All that's required is that you apply the programming tools and conventions presented so far in this course.

Apply the "string buffer method" for any numeric input. This instruction appears specifically in each of the assignments below, but in future labs, be sure to apply the method *always* unless specified otherwise.

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 **lab4** folder provided.

LAB 4a: Typewriter Graphics `[ Fill.cpp ]`
Write **Fill.cpp** to solve exercise 5.23 on page 225 of Deitel (reproduced below, with 5.22):


erwise.

5.22 *(Square of Asterisks)* Write a function that displays at the left margin of the screen a solid square of asterisks whose side is specified in integer parameter `side`. For example, if `side` is `4`, the function displays the following:

```
****
****
****
****
```

5.23 *(Square of Any Character)* Modify the function created in Exercise 5.22 to form the square out of whatever character is contained in character parameter `fillCharacter`. Thus, if `side` is `5` and `fillCharacter` is `#`, then this function should print the following:

```
#####
#####
#####
#####
#####
```

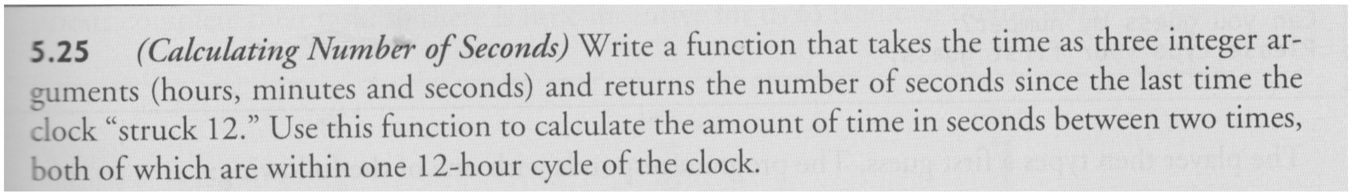
In `main`, read the side and fillCharacter using console input. Be sure to use understandable prompts, and remember to use the "string buffer method" to read the "size". Send them "by value" to the function, named as you wish. Allow for a range of 1-20 for side, and say it in your prompt so that the user knows what to expect. Check the side value, and if it's outside the 1-20 expected range, output no square.

Do use a prototype for the function. If you do this right, you should have a "void function" with two "parameters" -- an int and a char.

Apply the same enhancement that you did in lab 2a: use a sentinel-controlled loop to run the program over and over again until the user wants to quit. Quit when the user inputs an uppercase or lowercase Q for "size". Quit by breaking out of the loop -- do not `return` from inside the loop. If the user enters a value outside the 1-20 range, including anything that `atoi` resolves to zero, skip to the next cycle.

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

LAB 4b: Multi-parameter Functions `[ Seconds.cpp ]`
Write **Seconds.cpp** to solve exercise 5.25 on page 225 of Deitel (reproduced below). Ignore the last sentence, starting "Use this function...":



5.25 *(Calculating Number of Seconds)* Write a function that takes the time as three integer arguments (hours, minutes and seconds) and returns the number of seconds since the last time the clock "struck 12." Use this function to calculate the amount of time in seconds between two times, both of which are within one 12-hour cycle of the clock.

"Struck 12" means that if the time entered is 13:00:00, that's *one hour* past 12. 25:00:00 is also one hour past the last 12. Don't worry about range-checking any inputs: 70 for minutes is just 70 minutes.

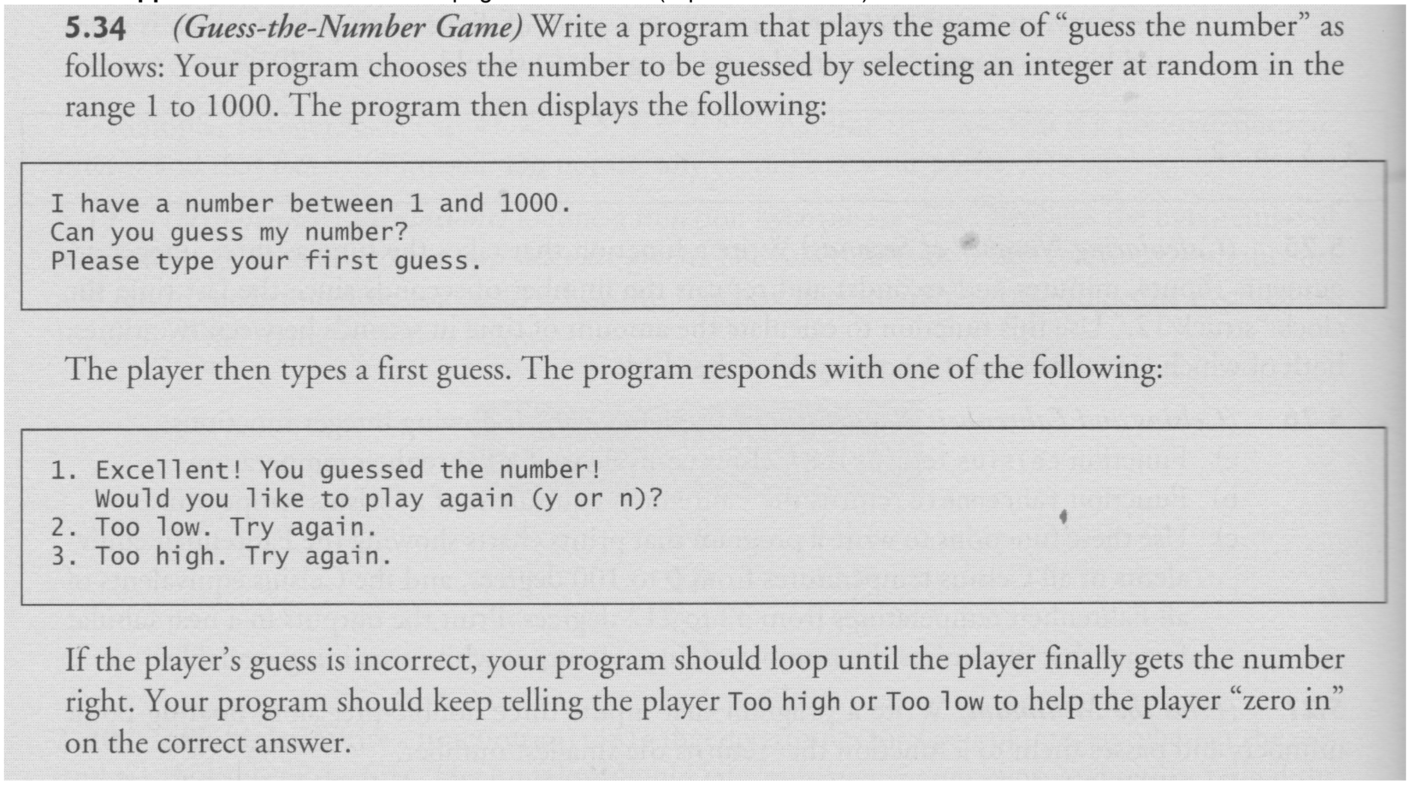
In `main`, read the hours, minutes, and seconds, using console input. Allow all three inputs on the same line of input, space-separated, by leaving out `cin.ignore`. Be sure to use understandable prompts, and remember to use the "string buffer method".

Do use a prototype for the function. If you do this right, you should have a "value-returning function" with three int "parameters".

Apply the same enhancement as in lab 4a, using a replay loop, quitting upon a q or Q for *any* of the inputs. That is, 1 for hour, 30 for minutes, and Q for seconds -- quit. 1 for hours, Q for minutes, quit -- *without waiting for seconds to be entered*! Q for hours, quit -- *without waiting for minutes to be entered*!

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

LAB 4c: Tabular Formatting `[ Guess.cpp ]`
Write **Guess.cpp** to solve exercise 5.34 on page 226 of Deitel (reproduced below):



5.34 *(Guess-the-Number Game)* Write a program that plays the game of "guess the number" as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1 to 1000. The program then displays the following:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
```

The player then types a first guess. The program responds with one of the following:

```
1. Excellent! You guessed the number!
   Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.
```

If the player's guess is incorrect, your program should loop until the player finally gets the number right. Your program should keep telling the player `Too high` or `Too low` to help the player "zero in" on the correct answer.

In `main`, prompt the user about "would you like to play again", and allow uppercase or lowercase Y and N as the answers. You decide how to handle input validation -- that is, what to do if the user enters something other than Y or N.

Include at least TWO functions, not counting main. You decide what gets done in the functions and what gets done in `main`, as long as the logic for determining if the guess is high, low, or correct, is in a function. The logic for repeating guesses until the right number is guessed may be either in main or in a function. This has a replay loop, allowing new numbers to be guessed. Put that loop in `main`, and not in the function.

*Post **Guess.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.