1. Consider the following program to print even numbers up to some user specified amount.

```cpp
#include <iostream>
using namespace std;

int main() {
    int stopValue;
    int counter;
    cout << "Enter a stopping value: ";
    cin >> stopValue;
    counter = 0;
    while (counter != stopValue) {
        cout << counter << endl;
        counter = counter + 2;
    } // end while
    cout << "Done" << endl;
} // end main
```

a) In the box, show a few lines of the programs output assuming the user enters 8 for a stopping value.

b) What would happen if the user enters 9 for a stopping value?

c) What modification would improve this program?

2. An infinite loop is one that would loop forever. (FYI, at a console window ctrl-c (^c) can be used to kill the running program. Debug | Stop Debugging if you are running via Debug | Start Debugging) Most infinite loops are caused by programmer error, but sometimes they are intentional. The following code uses an infinite loop and a break statement that immediately causes control to exit the loop.

```
// Program to calculate the average score using an infinite loop and break statement
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    int numberOfScores;
    double totalOfScores, score, average;
    totalOfScores = 0.0;
    numberOfScores = 0;

    while (true) {
        cout << "Enter a score (or -999) to exit: ";
        cin >> score;
        if (score < 0) {
            break; // drop out of loop
        } // end if
        totalOfScores = totalOfScores + score;
        numberOfScores++;
    } // end while
    average = totalOfScores / numberOfScores;
    cout << "The average is " << setprecision(1) << fixed << average << endl;
} // end main
```

Draw a flow chart for this code.
3. A `continue` statement also alters execution of a loop body. When a `continue` statement is encountered, the next iteration of the loop (including the `while` condition check) is immediately executed. Armed with this limited knowledge, trace the following program to determine its output.

```cpp
#include <iostream>
using namespace std;

int main() {
    int counter;
    for (counter = 1; counter <= 10; counter++) {
        if (counter % 3 == 0) {
            continue;
        // end if
        }
    cout << counter << endl;
    } // end for
} // end main
```

4. a) Complete the following loop which validates the menu choice “grade.” You only need to add a `continue` statement and a `break` statement.

```cpp
#include <iostream>
using namespace std;

int main() {
    const int SIZE = 50;
    char grade[SIZE];

    while (true) {
        cout << "Math Tutor Menu\n\n";
        cout << "1. First Grade\n";
        cout << "2. Second Grade\n";
        cout << "3. Third Grade\n";
        cout << "Enter your grade (1-3): ";
        cin.getline(grade, SIZE);
        if (!(strcmp(grade, "1") == 0
            || strcmp(grade, "2") == 0
            || strcmp(grade, "3") == 0)) {
            cout << "Invalid grade! Enter only 1, 2, or 3\n\n";
            continue;
        } // end if
    cout << "Valid grade entered \n"
    } // end while
} // end main
```

b) In the above code, why is reading the `grade` as a “string” better than reading it an integer?
5. In algebra, reading a “story problem” and being able to “solve for x,” is a difficult skill to learn. The analogy in Computer Science (IMHO) is reading a “story programming problem” and identifying the looping structure (and other control structures like if-statements). To get some more practice, let’s think about ways to print the following patterns of asterisks that scale in size based on a user’s input.

a) size of 9
```
*
**
***
****
*****
******
*******
********
*********
```

b) size of 9
```
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
```

c) size of 9
```
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
```

d) size of 9
```
*       *       *
*       *       *
*       *       *
*       *       *
*       *       *
*       *       *
*       *       *
*       *       *
*       *       *
```