Question 1: A Mysterious Program

When the Visual Attention Lab at UMB moved to its new place, the lab team discovered an ancient C++ program written on an old, torn sheet of paper (see next page). Can you help the lab team to find out what each of the following functions/variables of the program does and what the program’s output looks like? Use English words for your descriptions; you can also draw a schematic sketch of the output if you cannot describe it (but do not draw the entire output!).

Mystery::char a[size][size]: (3 points)

This 50×50 character array serves as a canvas for the program to draw on. Initially it is blank (filled with dots), then the program draws Xs on it and finally prints the array on the screen.

Mystery::int x, y: (3 points)

(x, y) indicates the current position of the drawing cursor on the canvas. It is updated whenever an X is drawn.

Mystery::Mystery(int x0, int y0): (4 points)

This is the constructor for the Mystery class. It initializes the canvas by completely filling it with dots. Moreover, it sets the initial cursor position to (x0, y0).

Mystery::Draw(direction d, int length): (4 points)

Starting at position (x, y), this function draws a straight line of Xs on the canvas. Direction d specifies the direction (up, down, left, or right) and int length specifies the maximum length of the line. The line drawing also terminates if the cursor leaves the canvas. In that case, the function returns false, otherwise it returns true.

Mystery::GenerateOutput(): (4 points)

This function draws a sequence of lines in the cyclic order of directions left, up, right, and down. The first line is of length one, and each following line is one X longer than the preceding one. The function terminates as soon as the drawing cursor leaves the canvas.

What does the output look like? (6 points)
It looks like a spiral of Xs starting in the center of the canvas, as shown below, but extending to size 50×50:

..........
.XXXXXXXX.
.X...X..X.
.X.XXX.XX.
X.X.X.X.X.
.X.XX.X.X.
.X.X..X.X.
.X X.XX..X.
.X .XXXX.X.
     .X.X.

**Question 2: Chase the Bugs!**

For his latest research project, Professor P. wants to write a program that sorts an integer array containing ten elements. Going for the simplest solution, he decides to implement a bubble sort algorithm. The basic idea of bubble sort is that you first compare the first two elements in the list (elements 1 and 2). If element 1 is greater than element 2, you swap the values of the two elements, that is, element 1 receives the value of element 2 and vice versa. If element 1 is less than or equal to element 2, nothing happens. You then do the same thing with elements 2 and 3, then elements 3 and 4, and so on, until elements 9 and 10. This way the greatest element in the list will be carried to the end of the list. You repeat the whole procedure, but this time you can stop after comparing elements 8 and 9, because you know that element 10 is in the desired position already. Next time you can stop after elements 7 and 8, and so on, until the whole list is sorted in ascending order.

As usual, Professor P. is in a hurry and seriously confused, so the first version of his program contains a lot of bugs. Please help him correct all the bugs so that the program creates the following desired output:

-3000
-54
0
1
16
22
34
512
1432
13245

Please indicate and correct all bugs in the program listing on the next page. Try to use as few changes as possible to make the program compile and run correctly. You may also insert and delete lines if you like to. Do not rewrite entire lines of code but use C++ syntax to keep the changes small.

(Of course this task is difficult to do without using a compiler. Try to find as many bugs as possible; you will receive bonus points if you actually manage to correct all the bugs.)
```cpp
#include <iostream>
#include <vector>
using namespace std;

void SwapIntegers(int &a, int &b)
{
    int temp = a;
    a = b;
    temp = b;
}

void BubbleSort(vector<int> &intVector)
{
    for (int i = intVector.size() - 1; i > 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            if (intVector[j] < intVector[j + 1])
            {
                SwapIntegers(intVector[j], intVector[j + 1]);
            }
        }
    }
}

int main()
{
    int intArray[] = {34, 1432, 1, -54, 16, 22, 13245, 512, -3000, 0};
    vector<int> intVector(intArray, intArray + 9);
    BubbleSort(&intVector);  // delete: '&'  
    <
    for (int i = 0; i <= intVector.size(); i++)
    {
        cout << intVector[i] << endl;
    }
    return 0;
}  // << <<

2 points for every identified bug, and -1 point for every newly introduced bug.
```
Question 3: Being Agile

Please describe what Agile software development is.

From Wikipedia:

Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams. It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.
Question 4: Technical Debt

In software development, what is technical debt and what problems can it cause?

From Wikipedia:

Technical debt (also known as design debt or code debt) is "a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution".

Technical debt can be compared to monetary debt. If technical debt is not repaid, it can accumulate 'interest', making it harder to implement changes later on. Unaddressed technical debt increases software entropy. Technical debt is not necessarily a bad thing, and sometimes (e.g., as a proof-of-concept) technical debt is required to move projects forward. On the other hand, some experts claim that the "technical debt" metaphor tends to minimize the impact, which results in insufficient prioritization of the necessary work to correct it.

As a change is started on a codebase, there is often the need to make other coordinated changes at the same time in other parts of the codebase or documentation. The other required, but uncompleted changes, are considered debt that must be paid at some point in the future. Just like financial debt, these uncompleted changes incur interest on top of interest, making it cumbersome to build a project. Although the term is used in software development primarily, it can also be applied to other professions.