Lab Questions

1. Write a function `str_to_int()` that converts a `string` representing a signed integer (i.e., a string consisting of a possible plus or minus sign followed by one or more digits) to its corresponding numeric value. The function should take a single parameter that is a string of digits and return an **`int`**. If the `string` is not formatted correctly, the value zero should be returned.

   Write a program to test the `str_to_int()` function. Also, test the `str_to_int()` function with the program below.

```
int main() {
    string s;
    while (cin >> s) {
        cout << str_to_int(s) << endl;
    }
}
```

2. We'll say that an element in an array is *alone* if there are values before and after it, and those values are different from it. Write a function `notAlone()` that return a version of the given array where every instance of the given value which is alone is replaced by whichever value to its left or right is larger.

   - `notAlone({1, 2, 3}, 2)` returns `{1, 3, 3}`
   - `notAlone({1, 2, 3, 2, 5, 2}, 2)` returns `{1, 3, 3, 5, 5, 2}`
   - `notAlone({3, 4}, 3)` returns `{3, 4}`

3. Write a function `scoresAverage()` that given a `vector<int>` of scores, compute the integer average of the first half and the second half, and return whichever is larger. We'll say that the second half begins at index `length/2`. The vector length will be at least 2.

   To practice decomposing a program into simple functions, write a separate helper method

```
int average(const vector<int>& a, int start, int end)
```

   which computes the average of the elements between indexes `start..end`. Call your helper method twice to implement `scoresAverage()`. Normally you would compute averages with doubles, but here we use **`int`**s so the expected results are exact.

   - `scoresAverage({2, 2, 4, 4})` returns 4.
   - `scoresAverage({4, 4, 4, 2, 2, 2})` returns 4.
   - `scoresAverage({3, 4, 5, 1, 2, 3})` returns 4.

4. *(Calculating the Value of π)* Calculate the value of π from the infinite series

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

   Print the value of π approximated by computing the first 200,000 terms of this series. How many terms do you have to use before you first get a value that begins with 3.14159?

   Write a function **double** `pi(`**int** `n)` that returns the value of π approximated by computing the first n terms of the above series.

5. Write a program that randomly fills in 0s and 1s into a 4-by-4 matrix, prints the matrix, and finds the first row and column with the most 1s. Here is a sample run of the program:

```
0 0 1 1
0 0 1 1
1 1 0 1
1 0 1 0

The largest row index: 2
The largest column index: 2
```

   Implement and use following functions in your program:

```
// generate a random matrix of size m x n
vector<vector<int>> genRandomMatrix(int m, int n)

// print a matrix
void printMatrix(const vector<vector<int>>& a)

// find the row index with the most 1s
int findMaxOnesRow(const vector<vector<int>>& a)

// find the column index with the most 1s
int findMaxOnesCol(const vector<vector<int>>& a)
```

   Note that the matrix is a `vector` of `vectors`. It can be created by `vector<vector<int>> a(m,vector<int>(n)`
   The number of rows of `a` is `a.size()` and the number of columns is `a[0].size()`.

6. Nine coins are placed in a 3-by-3 matrix with some face up and some face down. You can represent the state of the coins using a 3-by-3 matrix with values 0 (heads) and 1 (tails). Here are some examples:

```
0 0 0       1 0 1       1 1 0       1 0 1       1 0 0
0 1 0       0 0 1       1 0 0       1 1 0       1 1 1
0 0 0       1 0 0       0 0 1       1 0 0       1 1 0
```

   Each state can also be represented using a binary number. For example, the preceding matrices correspond to the numbers

```
000010000        101001100        110100001        101110100        100111110
```

There are a total of 512 possibilities, so you can use integer $0, 1, 2, 3, \ldots, 511$ to represent all states of the matrix. Write a program that prompts the user to enter a number between 0 and 511 and displays the corresponding matrix with the characters `H` and `T`. In the following sample run, the user entered 7, which corresponds to `000000111`. Substituting `H` for 0 and `T` for 1, the output is following.

```
Enter a number between 0 and 511: 7
H H H
H H H
T T T
```

Implement and use following functions in your program:

```
// convert an integer between 0 and 511 to a 3x3 binary matrix
vector<vector<int>> toMatrix(int x)

// print a matrix
void printCoinsMatrix(const vector<vector<int>>& a)
```

7. Given a 9-by-9 array of integers between 1 and 9, write SudokuCheck.java to check if it is a valid solution to a Sudoku puzzle: each row, column, and block should contain the 9 integers exactly once.

```
 5 3 4 | 6 7 8 | 9 1 2
 6 7 2 | 1 9 5 | 3 4 8
 1 9 8 | 3 4 2 | 5 6 7
-------+-------+------
 8 5 9 | 7 6 1 | 4 2 3
 4 2 6 | 8 5 3 | 7 9 1
 7 1 3 | 9 2 4 | 8 5 6
-------+-------+------
 9 6 1 | 5 3 7 | 2 8 4
 2 8 7 | 4 1 9 | 6 3 5
 3 4 5 | 2 8 6 | 1 7 9
```

The above Sudoku board can be stored in a 2d vector as follows:

```
vector<vector<int>> a = {
{5, 3, 4, 6, 7, 8, 9, 1, 2},
{6, 7, 2, 1, 9, 5, 3, 4, 8},
{1, 9, 8, 3, 4, 2, 5, 6, 7},
{8, 5, 9, 7, 6, 1, 4, 2, 3},
{4, 2, 6, 8, 5, 3, 7, 9, 1},
{7, 1, 3, 9, 2, 4, 8, 5, 6},
{9, 6, 1, 5, 3, 7, 2, 8, 4},
{2, 8, 7, 4, 1, 9, 6, 3, 5},
{3, 4, 5, 2, 8, 6, 1, 7, 9} };
```