Algorithms in C++: Assignment 7

1. Objective

Your goal is to solve the maximum sum descent problem.

2. Problem

You are given a triangular-shaped pile of integers in a file. There may be anywhere from 1 to 100 rows in the triangle. Your task is to find the maximum total from the top to the bottom, outputting the total as well as the values that comprise that total.

For the triangle below, the numbers in red are on the maximal path.

```
75
95 64
17 42 82
18 35 87 10
```

The max sum is 75 + 64 + 82 + 87, which is 308. The program takes in a single command line argument, which is the name of the file containing the integers. All integers are guaranteed to be no more than 2 digits, so the sum will definitely fit into an integer.

The output will look as follows for a given execution. First the table is displayed, then the max sum is displayed, followed by the values that comprise that sum. See below:

```
$ ./maxsumdescent input.txt
75
95 64
17 47 82
18 35 87 10
20  4 82 47 65
19  1 23 75  3 34
88  2 77 73  7 63 67
99 65  4 28  6 16 70 92
41 41 26 56 83 40 80 70 33
41 48 72 33 47 32 37 16 94 29
53 71 44 65 25 43 91 52 97 51 14
70 11 33 28 77 73 17 78 39 68 17 57
91 71 52 38 17 14 91 43 58 50 27 29 48
63 66  4 68 89 53 67 30 73 16 69 87 40 31
 4 62 98 27 23  9 70 98 73 93 38 53 60  4 23
Max sum: 1074
Values: [75, 64, 82, 87, 82, 75, 73, 28, 83, 32, 91, 78, 58, 73, 93]
```

The program reads in a 2D array of positive integers from the file whose name is supplied on the command line. The array will look like the one above; that is, the first row has 1 integer, the second, 2, and so on.

You must write code in the following functions:

- `void display_table()`
- `int compute_max_sum()`

- `vector<int> backtrack_solution()`
- `bool load_values_from_file(const string &filename)`
- `void cleanup()`
- `int main(int argc, char * const argv[])`

3. Advice

Your program should work for a blank file as well as any file with [1, 100] rows. The max sum of a blank file is 0, with the values being the empty list []. Check your sum by looping over the values in the vector. It should match the sum returned in `compute_max_sum()`. `backtrack_solution()` should run in $\theta(n)$ where n is the number of rows in the triangle. A good dynamic programming / backtracking implementation will be very efficient, and on Linux Lab, should easily meet the 0.2 second requirement even without enabling compiler optimizations. If you find your program running too slowly, make sure you are not doing unnecessary computations.