Ben Bitdiddle wants to optimize the following piece of working code:

```
/* The elements of A and B are known to be nonnegative and not
   aliased */
/* k is known only at runtime */
static const int N = (1 << 30);
for (int i = 0; i < N; i++) {
  A[i] = k * B[i];
}
```

Using what he learned in this class, he rewrites the code as:

```
/* The elements of A and B are known to be nonnegative and not
   aliased */
/* k is known only at runtime */
static const int N = (1 << 30);
A[N - 1] = -1;
int *a = A , *b = B;
while (*a >= 0) {
  *(a++) = k * *(b++) ;
}
*a = k * *b;
```

Ben compiles both codes with GCC using optimization `-O3`. When he runs the two codes, however, Ben finds that his new code is several times slower than his old code because it no longer vectorizes. Why does it fail to vectorize?

Solution:

There are two good reasons why this code doesn't vectorize:

1. The for loop had a set number of iterations that the compiler could easily vectorize, but the while loop doesn't know when to stop.
2. The values of a and b change with each iteration, so there is backwards dependency.

Given a matrix `M` in row-major format, we wish to compute the row sums (i.e., the sums across the rows).

```
 1 #define ROWS 2048
 2 #define COLS 3072
 3
 4 void compute_row_sums(int M[ROWS][COLS], int row_sums[ROWS]) {
 5   for (int i = 0; i < ROWS; i++) {
 6     row_sums[i] = 0;
 7     for (int j = 0; j < COLS; j++) {
 8       row_sums[i] += M[i][j];
 9     }
10   }
11 }
```

Can the code in `compute_row_sums()` be transformed to use vector hardware effectively? If yes, explain briefly where and how to vectorize `compute_row_sums()`. If no, explain briefly why the function cannot be vectorized.

Solution:
The outer for loop can't be vectorized very easily, but the inner one can be vectorized using strip mining. We could create four (or eight, depending on the vector hardware) lanes and do something to the effect of

```
for (int j = 0; j < COLS; j += 4) {
  for (int k = 0; k < 4; k++) {
    row_sums[i][k] += M[i][j+k];
  }
}
```

The innermost for loop will then become a vector add.