1. The following loops are executed on an Intel CPU with AVX SIMD instructions. These instructions operate on 256-bit *superwords*. Examine each of the following codes and identify if it is vectorizable or not. That is, can the loop be executed in parallel using SIMD methods? If not, state why and suggest way could be changed to permit vectorization.

   a.
   ```
   double x[16], y[16];
   for (int j = 1; j < 16; ++j)
      x[j] = x[j-1] + y[j];
   ```

   No. Read-after-write (RAW) data dependency prevents vectorization. This recursion is difficult to do in parallel. The parallel version of this is called a *prefix scan* (or prefix sum in this case) and is a standard parallel algorithm using binary recursion.

   b.
   ```
   double x[16], y[16];
   for (int j = 0; j < 16; ++j)
      x[j/2] = x[j] + y[j];
   ```

   No. Multiple iterations within the vector width (i.e., lanes) write to the same array address. When executed serially, at i=0, x[0] updates itself but at i=1, x[0] depends up x[1]. And at i=2, x[1] depends upon x[2] and then x[3]. I can't think of any way to do this in parallel.

   c.
   ```
   double x[16], y[16];
   for (int j = 0; j < 16; ++j)
      x[j] = x[j] + y[j];
   ```

   Yep!

   d.
   ```
   void f (int n, float *x, float *y, float *z, float alpha)
   {
     for (i = 0; i < n; i++)
       z[i] = x[i] + alpha * y[i];
   }
   ```

   Probably not due to the potential for array (pointer) aliasing. This can be fixed by (i) using 'restricted' pointers (not standard in C++); (ii) adding a #pragma before the for loop such as `#pragma ivdep` (not standard) or `#pragma omp simd` (standard within OpenMP 4.0+).

   e.
   ```
   double sum = 0;
   for (ptr = head; ptr != NULL; ptr = ptr->next)
      sum = sum + ptr->value;
   ```

   Nope. The loop bound is not a run-time constant (i.e., not known before the loop begins). Also, there is no way to determine multiple ptr->value's in parallel as the linked list forces sequential access.

   f.
   ```
   float x[8], y[128];
   for (int i = 0; i < 128; ++i)
   ```

```
y[ i ] = y[ i ] + x[ i%8 ];
```

Yes. This is an odd structure but there is nothing to preclude SIMD vectorization: known loop count, independence between iterations.