Question 6 - Vectors and Pointers

Below, I have three header files that specify the types **L** and **LVec**, and the prototype for a procedure **print_LVec()**. An **LVec** is a nested data structure with vector at the top level. The vector contains either lists or pointers to lists, and the lists contain either doubles or pointers to doubles. The procedure **print_LVec()** is supposed to print out all of the doubles, one per line.

header-1.h

```
typedef list <double *> L;
typedef vector <L> LVec;

void print_LVec(LVec &v);
```

header-2.h

```
typedef list <double> L;
typedef vector <L *> LVec;

void print_LVec(LVec &v);
```

header-3.h

```
typedef list <double *> L;
typedef vector <L *> LVec;

void print_LVec(LVec &v);
```

Below are six implementations of **print_LVec()**. For each of these implementations, and each header file above, choose one of the multiple choice answers below about how the implementation compiles and runs with the header file:

1. The implementation compiles correctly with the header file, and they print the doubles correctly without making any extraneous copies of the lists.
2. The implementation/header compile and print correctly, but they make extra copies of the lists.
3. The implementation/header compile correctly, but they print something other than the doubles.
4. The implementation/header do not compile correctly because there are problems in the **for** loops.
5. The implementation/header do not compile correctly because there is a problem that is not in a for loop.

```
void print_LVec(LVec &v)    // Implementation A
{
  int i;
  L::iterator lit;
  L *lp;

  for (i = 0; i < v.size(); i++) {
    lp = v[i];
    for (lit = lp->begin(); lit != lp->end(); lit++) {
      cout << *lit << endl;
    }
  }
}
```

```
void print_LVec(LVec &v)    // Implementation B
{
  int i;
  L::iterator lit;

  for (i = 0; i < v.size(); i++) {
    for (lit = v[i].begin(); lit != v[i].end(); lit++) {
      cout << *(*lit) << endl;
    }
  }
}
```

```
void print_LVec(LVec &v)    // Implementation C
{
  int i;
  L::iterator lit;

  for (i = 0; i < v.size(); i++) {
    for (lit = v[i]->begin(); lit != v[i]->end(); lit++) {
      cout << *(*lit) << endl;
    }
  }
}
```

```
void print_LVec(LVec &v)    // Implementation D
{
  int i;
  L::iterator lit;

  for (i = 0; i < v.size(); i++) {
    for (lit = v[i]->begin(); lit != v[i]->end(); lit++) {
      cout << *lit << endl;
    }
  }
}
```

```
void print_LVec(LVec &v)    // Implementation E
{
  int i;
  L::iterator lit;
  L *lp;

  for (i = 0; i < v.size(); i++) {
    lp = v[i];
    for (lit = lp.begin(); lit != lp.end(); lit++) {
      cout << *lit << endl;
    }
  }
}
```

```
void print_LVec(LVec &v)    // Implementation F
{
  int i;
  L::iterator lit;
  L lp;

  for (i = 0; i < v.size(); i++) {
    lp = v[i];
    for (lit = lp.begin(); lit != lp.end(); lit++) {
      cout << *(*lit) << endl;
    }
  }
}
```