1.

```
template<typename T, typename int rows = 1, typename int cols=1>
auto make2DArray(const T &init)
{
/* Version 1:
   std::array<std::array<T, cols>, rows> array2D; // define a 2D array
   // then initialize to the init value using loops
*/
   // Version 2:
   using RowType = std::array<T, cols>;
   std::array<RowType, rows> array2D;

   RowType InitArrayRow;          // a temporary array representing a row
   InitArrayRow.fill(init);       // fill InitArrayRow with init
   array2D.fill(InitArrayRow);    // fill array2D with InitArrayRow

   return array2D;
}
```

2.

```
   cin >> std::noskipws;
   std::istream_iterator<char> start(cin), finish;
   std::ostream_iterator<char> out(cout, " ");
   std::copy(start, finish, out);
```

3.

1.

```
void sortVector(std::vector <int> & vec)
{
   std::multiset<int> tempMultiset(vec.begin(), vec.end());
   // Empty the vector, then fill it with the elements from the multiset.
   vec.clear();
   vec.insert(vec.begin(), tempMultiset.begin(), tempMultiset.end());
}
```

2.

```
void sortVector(std::vector <int> & vec)
{
   std::sort(vec.begin(), vec.end());
}
```

4.

```
template<class T>
class AlternateSum
{
   int sign;
public:
   AlternateSum(int s=-1) : sign(s){}
   T operator()(const T &obj1, const T &obj2)
   {
      sign = -sign;
      return obj1 + (obj2 * sign);
   }
};

int AltSum(vector<int> & v)
{
   return accumulate(v.begin(), v.end(), 0, AlternateSum<int>());
}
```

5.

```
std::vector<int> MakeAndFillVector(int n, int x, int y)
{
   // create, fill, and return a vector filled with
   // n random integers between x and y, inclusive.
   // swap x and y if x > y
   auto random = [x, y]()mutable  // lambda captures x and y value;
   // hence must use mutable so that this lambda can modify x and y
   {
      if (y < x) std::swap(x, y); // possibly modifying x and y
      return std::rand() % (y - x + 1) + x;
   };
   std::vector<int> vec; // an empy vector
   std::generate_n(std::back_inserter(vec), n, random); // must use an inserter
   return vec;
}
```