1. Odometer problem
   1. Car with odometer that overflows at 9999
      1. Goal: print every setting of the odometer
   2. Properties
      1. odom(-see notes-)
      2. //10 recursive calls
   3. Code
      1. //array = odometer
      2. //len = length of array
      3. //k = # of filled slots
      4. void(odometer, int\* array, int len, int k){
         1. if (k == len) {
            1. print(array, len);
            2. return;
         2. }
         3. int i;
         4. //Try each possible value in our first empty slot k
         5. for (i = 0; i < 10; i++){
            1. //Skip over any choice where the item equals what is in the previous slot
            2. if (k != 0 && i == array[k-1])

continue;

* + - * 1. //Putting i in slot k
        2. array[k] = i;
        3. odometer(array, len, k+1);
      1. }
    1. }
    2. void print(int\* array, int len) {
       1. int i;
       2. for (i = 0; i < len; i++)
          1. printf(“%d”, array[i]);
       3. return;
    3. }

1. Review on continue and break
   1. continue: finishes an iteration and goes on to the next.
   2. break: breaks out of loop.
2. Printing out subsets
   1. Example: sets of {0, 1, 2} as binary
      1. {} 🡪 000
      2. {0} 🡪001
      3. {1} 🡪010
      4. {2} 🡪100
      5. {0,1} 🡪011
      6. {0,2} 🡪101
      7. {2,1} 🡪110
      8. {0,1,2} 🡪111
3. Permutations
   1. Like an odometer, but you can’t reuse a digit.
   2. Example
      1. 0, 1, 2
      2. 0, 2, 1
      3. 1, 0, 2
      4. 1, 2, 0
      5. 2, 0, 1
      6. 2, 1, 0
   3. Problem: array of 0, 1, 2, 3, 4, 5, 6, 7
      1. Array partially filled in:
         1. Array[8] = {3, 1, 6};
         2. //k = # of unfilled
         3. int k = 3;
      2. used\_array as an array of which numbers have been used
         1. used\_array[8] = {0, 1, 0, 1, 0, 0, 1, 0}
         2. for (i = 0; i < n; i++) {
            1. if (!used\_array[i]) {

array[k]] = i;

used\_array[i] = i;

permutation (array, used\_array, k+1, n);

* + - * 1. }
      1. }
    1. Permutation function
       1. permutation(array, used\_array, int k, int len) {
          1. if (k == len) {

//Process permutation

print(array, len);

return;

* + - * 1. }
        2. int i;
        3. //Try each possible option in slot k
        4. for (i = 0; i < len; i++) {

//Skip anything that has been used before

if (!used\_array[i]) {

array[k] = i;

used\_array[i] = i;

permutation(array, used\_array, k+1, len);

used\_array[i] = 0;

}

* + - * 1. }
      1. }

1. Derangements
   1. Derangement: permutation where no integer is in its original place.