Q1. Write a C++ function

```
void printOdds(int num) { . . . }
```

which prints a list of all odd numbers between 0 and the given positive number num, with a comma printed after each number. For example, calling your method from main() with:

```
printOdds(11);
```

should result in the screen output:

```
1, 3, 5, 7, 9, 11,
```

Q2. Write a C++ function

```
int countOdds(int arr[], int size) { . . . }
```

which returns the count of odd integers that are contained in the array arr. The size parameter is the length of the array. For example, when you test your function using the following code in main()

```
int arr[] = {3, 9, 4, 6, 7, 8, 1};
cout << "Number of odds = " << countOdds(arr, 7) << endl;
```

the output printed should be:

```
Number of odds = 4
```

Q3. Write a C++ function

```
void getAndPrintNameChars() { . . . }
```

which prompts the user for their name and then prints out the characters of their name, with each character numbered and printed on a separate line. For example, if the user inputs “Robert”, the following should be printed:

```
1. R
2. o
3. b
4. e
5. r
6. t
```

Use std::cin and the C++ string class in your implementation.

Q4. Write a method

```
   double frac(double d) { . . . }
```

to return the fractional part of a double value. This should require only one line of code and needs no maths library functions. Hint: use casting or automatic type conversion to do this in a single line of function code.

Example test code:

```
   cout << frac(4.32) << endl;  // should print the value 0.32
```

Q5. Using only basic C/C++ features, loops, bit shifting and bit wise operators, write a function

```
   void printBinary(int a) { . . . }
```

which prints out the binary value of a given int value, printing a space between each group of 8 bits. For example, calling the function as

```
   printBinary(1234);
```

should result in the printout:

```
00000000 00000000 00000100 11010010
```

Q6. Write a function

```
   void printArray(int arr[], int size) { . . . }
```

which prints each element in the array arr with each element followed by a comma. The size parameter specifies the number of elements in the array passed to the function. For example, the following test code in main():

```
   int arr[] = {3, 9, 4, 6, 7, 8, 5};
   printArray(arr, 7);
```

should result in the printout:

```
3, 9, 4, 6, 7, 8, 5
```

Q7. Write a function

```
void printMemory(int[] mem, int size) { . . . }
```

which prints out the memory address (in decimal format) and the value of each element in a given integer array. For example, when tested with the code:

```
int arr[] = {3, 9, 4, 6, 7, 8, 1};
printMemory(arr, 7);
```

The printout should appear similarly to the following:

```
Address     Contents
7208660     3
7208664     9
7208668     4
7208672     6
7208676     7
7208680     8
7208684     1
```

Q8. Write a function

```
void printNumberPattern(int n) { . . . }
```

that takes an integer n>=1 and prints a triangle pattern similar to below (n=9 for this example):

```
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
4 5 6 7 8 9
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```

Hint: this can be done using one for loop nested inside another for loop.

Q9. Write a method

```
   void putLargestFirst(double a[], int size) { . . . }
```

which takes an array or doubles of the given size, and swaps the largest element with the first element in the array. For example, the following test code:

```
   double arr[] = {3.1, 9.2, 4.0, 6.7, 12.1, 8.9, 1.4};
   putLargestFirst(arr, 7);
```

should result in the array elements being rearranged as:

```
{12.1, 9.2, 4.0, 6.7, 3.1, 8.9, 1.4}
```

Q10. Write a function

```
   bool isUnique(int a[], int size) { . . . }
```

which returns true if there are no repeated values (all values unique) in the array and returns false otherwise.

Hint: use one for loop nested inside another for loop