1. *Matrix addition:* Write a program *matrix_ add.cpp* to add two matrices a, b of the same size and print the result. You don't need to take input from command-line or standard-input. Input matrices can be initialized as follows.

```
int a[][4] = { {2, 5, 7, 1},
               {1, 3, 6, 1},
               {5, 4, 1, 3} };
```

```
int b[][4] = { {1, 9, 5, 0},
               {7, 1, 5, 4},
               {3, 4, 2, 8} };
```

2. *Column sum:* Given a 2d-array `a[][]` and an index `j`, write a program `sum_col.cpp` that compute sum of elements in column `j` of `a[][]`. You don't need to take input from command-line or standard-input.

3. *Matrix transpose:* Write a program `transpose.cpp` that compute the transpose of matrix given as 2d-array `a[][]`. Store the transpose in matrix `b[][]` and also print it. You don't need to take input from command-line or standard-input.

   Example: Following two matrices are transpose of each other.

```
{ {9, 5, 7},
  {1, 3, 6},
  {5, 4, 1},
  {0, 8, 2} }
```

```
{ {9, 1, 5, 0},
  {5, 3, 4, 8},
  {7, 6, 1, 2} }
```

4. *Upper-triangular matrix:* Write a program that checks whether a given square matrix is an upper-triangular matrix. A matrix is said to be an upper-triangular matrix if all values under the main diagonal are zeros. Example:

```
{ {1, 2, 3, 4},
  {0, 5, 6, 7},
  {0, 0, 8, 9},
  {0, 0, 0, 1} }
```

5. *Magic square:* Write a program called `is_magic_square.cpp` that given a two-dimensional array of integers, prints `true` if it is a magic square, and `false` otherwise.

   A square matrix is a magic square if it is square in shape (same number of rows as columns, and every row the same length), and all of its row, column, and diagonal sums are equal. For example, following two matrices are magic squares because all eight of the sums are exactly `15` and `65` respectively.

```
{ {2, 7, 6},
  {9, 5, 1},
  {4, 3, 8} }
```

```
{ { 1,  7, 13, 19, 25},
  {14, 20, 21,  2,  8},
  {22,  3,  9, 15, 16},
  {10, 11, 17, 23,  4},
  {18, 24,  5,  6, 12} }
```

6. *Checking an expression for 'well-bracketedness'* Read a string of characters representing a bracketed expression from a file one character at a time until end of file. Maintain a count of how many opening parentheses `'('` and closing parentheses `')'` have been encountered.

   Declare an integer variable `level`, initialized to zero. At each opening parenthesis, increase its value by one, and print its new value. At each closing parenthesis, decrease its value by one, and print its new value. If the level becomes negative, print `"Unmatched ')'"`, just once regardless of the number of unmatched `')'`s. At end-of-file, if the level is positive print `"Unmatched '('"`. If the level is zero and there have been no unmatched `')'`s then print `"Well-bracketed expression"`. E.g.:

   Input: `(a*(b-c)-((d/e)+f))`
   Output:

   ```
   Level 1
   Level 2
   Level 1
   Level 2
   Level 3
   Level 2
   Level 1
   Level 0
   Well-bracketed expression
   ```

   Note: Since `std::string` behaves like an array of `char`, you can iterate over the characters in a string `str` using the either the range-based `for` loop:

   ```
   for (char c : str) {
       // do something with c
   }
   ```

   or the traditional `for` loop:

   ```
   for (int i = 0; i < str.size(); i++) {
       char c = str[i];
       // do something with c
   }
   ```

   The `size()` method returns the length of the string.

7. *Smallest/Longest words:* Write a program that prints longest and smallest word in a given sentence. For example, if the input is

   ```
   Gedanken means thoughts in German. The term was popularized by Albert Einstein, who
   applied gedankenexperiment to his work conceptualizing the theory of relativity.
   ```

   then your program should print

   ```
   Smallest word: in
   Longest word: gedankenexperiment
   ```

8. *Longest run:* Write a program *longest_ run.cpp* that reads in a sequence of integers from standard input and prints out both the integer that appears in a longest consecutive run and the length of the run. For example, if the input is `1 2 2 1 5 7 7 7 7 2 2 2 1 1`, then your program should print

   ```
   Longest run:  4 consecutive 7s
   ```

9. *The Sieve of Eratosthenes:* This is a nice exercise that involves writing a program to generate all the prime numbers less than a given number n. (Remember that a prime number is a number whose only factors are 1 and itself or, put another way, is not exactly divisible by any numbers other than 1 and itself. So the first few primes are $2, 3, 5, 7, 11, 13, 17, 19, \ldots$)

   Only attempt the program if you understand Eratosthenes' sieve method as follows:

   1. Start with the first number not crossed off (i.e. 2). Call this k.
   2. Cross off (or 'sieve out') all the multiples of k that are less than or equal to n (since these are obviously not prime).
   3. Repeat steps 2 and 3 with the next number k that is not crossed off. Terminate when you have reached n.

   The prime numbers are all the numbers not sieved out.

   Using paper and pencil run Eratosthenes sieve by hand on the numbers 2..30.

   To write a program to carry out this method, use a `bool` array of size `N` with all elements initially set to `false`.

   Crossing off a number `k` involves setting the `k`th array element to `true`.

   Make your program print out the number of primes less than `N` as well as the primes themselves.

   How many prime numbers are there less than 1 million?

   Optional question: Can you make your program more 'efficient' by stopping one of your loops at the square root of `N`, rather than `N` itself? Explain why.

10. *Frequency table:* Read an input integer n from the keyboard. Generate `100,000` random integers in the range `1` to `n`. Use an array `A` to keep count of the frequency of occurrence of each integer (i.e. use `A[1]` to keep track of the number of occurrences of the integer `1`, `A[2]` for the integer `2` and so on). Print out the number of times each integer from `1` to `n` was generated.

    Question: what happens if the number input from the keyboard is larger than the size of the array `A`?

11. *Finding your sherbet:* A large number of IBA students are attending an Iftar party. Each guest is drinking a glass of Rooh-Afza sherbet. An emergency causes the lights to go out and the fire alarm to go off. The guests calmly put down their glass and exit the building. When the alarm goes off, they re-enter and try to retrieve their glass. However, the lights are still off, so each student randomly grabs a glass of Rooh-Afza. What are the chances that at least one student gets his or her original glass? Write a program `my_sherbet.cpp` that takes an input n and runs $1,000$ simulations this event, assuming their are n guests. Print the fraction of times that at least one guest gets their original glass. As n gets large, does this fraction approach 0 or 1 or something in between?

12. *Random permutation:* Write a program `permutation.cpp` so that it takes an input `N` and prints a random permutation of the integers `0` through `N-1`. Also print a checkerboard visualization of the permutation. As an example, the permutation `{2, 3, 1, 4, 0}` corresponds to:

```
* * * * Q
* * Q * *
Q * * * *
* Q * * *
* * * Q *
```

Hint: Use the shuffle algorithm discussed in the lectures to get a random permutation.

13. *8 queens checker:* A permutation of the integer 0 to $n-1$ corresponds to a placement of queens on an n-by-n chessboard so that no two queens are in the same row or column. Write a program `queens_checker.cpp` that determines whether or not a permutation corresponds to a placement of queens so that no two are in the same row, column, or diagonal. As an example, the permutation `{ 4, 1, 3, 0, 2 }` is a legal placement:

```
* * * Q *
* Q * * *
* * * * Q
* * Q * *
Q * * * *
```

Try to do it without using any extra arrays besides the length n input permutation `q`. Hint: to determine whether setting `q[i]` conflicts with `q[j]` for `i < j`.

- if `q[i]` equals `q[j]`: two queens are placed in the same row
- if `q[i] - q[j]` equals `j - i`: two queens are on same major diagonal
- if `q[j] - q[i]` equals `j - i`: two queens are on same minor diagonal