Input: ~~matrix~~ vector with coefficients of ~~objectives~~ function - C,
matrix of coefficients of constraints - A,
vector of right-hand side numbers - b.

Output: vector x with solution,
maximum value of function.

Generally, our implementation will be able to solve any maximization problem of this kind:

$F(x_1, x_2) = ax_1 + bx_2$

$\begin{cases} cx_1 + dx_2 \le e \\ fx_1 + gx_2 \le h \end{cases}$ for $x_1, x_2 \ge 0$

Example:

Lab 3 v1: maximize $F(x_1, x_2, x_3) = 9x_1 + 10x_2 + 16x_3$

subject to $\begin{cases} 6x_1 + 5x_2 + 4x_3 + s_1 = 120 \\ 3x_1 + 2x_2 + 4x_3 + s_2 = 96 \\ 5x_1 + 3x_2 + 3x_3 + s_3 = 180 \\ x_1, x_2, x_3, s_1, s_2, s_3 \ge 0 \end{cases}$

Input:

C: $\begin{bmatrix} 9 \\ 10 \\ 16 \end{bmatrix}$ A: $\begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 4 \\ 5 & 3 & 3 \end{bmatrix}$ b: $\begin{bmatrix} 120 \\ 96 \\ 180 \end{bmatrix}$

Elements in the implementation:

main_matrix: $\begin{bmatrix} 6 & 5 & 4 & 1 & 0 & 0 & 120 \\ 3 & 2 & 4 & 0 & 1 & 0 & 96 \\ 5 & 3 & 3 & 0 & 0 & 1 & 180 \end{bmatrix}$ (A | I | b)

basis: $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ (initially)

ratio: $\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$ (will be calculated) (initially could be $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$)

func_coef: $[\underbrace{9\ 10\ 16}_{c}\ 0\ 0\ 0]$

profit: $[0\ 0\ 0\ 0\ 0\ 0\ 0]$ (initially)

basis_el: $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ (initially, further there will be indexes of x in order of basis)

net_evaluation (or net_eval): $[\underbrace{9\ 10\ 16}_{c}\ 0\ 0\ 0]$ (initially)

Class Matrix

(2d-array of numbers)

+ multiplyBy(Vector): Vector
+ getEl(~~index~~ row, col): ~~number~~ float
+ setEl(~~index~~ row, col, Value): void
+ getRow(index): Vector
+ setRow(index, Vector): void
+ getCol(index): Vector
+ setCol(index, Vector): void
+ transpose(): matrix

Class Vector

(1-d array of numbers)

+ size(): ~~number~~ int
+ getEl(index): float
+ setEL(index, value): void
+ subtract(Vector): Vector
+ divideBy(float): Vector
+ multiplyBy(float): Vector

Code structure:

1. read C, A, b from input.
2. create and fill main_matrix, basis, basis_el, func_coef, profit, net_eval, ratio.
3. go through iterations:

```
for(int i=0; i<A.size(); i++){
    pivot_col = define_pivot_col(net_eval);
    ratio = calc_ratio(pivot_col, main_matrix);
    pivot_row = define_pivot_row(ratio);
    pivot_el = define_pivot_element(main_matrix,
                                    pivot_col, pivot_row);
    basis = define_basis(basis, pivot_row, pivot_col,
                                    func_coef);
    basis_el = define_basis_el(basis, pivot_row, pivot_col);
    main_matrix = updat_main_matrix(main_matrix,
                                    pivot_row, pivot_col, pivot_el);
    profit = calc_profit(main_matrix, basis);
    net_eval = calc_net_eval(func_coef, profit);
    if(check_net_eval) break;
```

4. Print answer:

```
for(int i=0; i<basis.size(); i++):
    print("x" + basis_el.getEL(i) + " = " + basis.getEL(i));
print("Profit = " + profit.getEl(profit.size()-1));
```

Functions:

1) define_pivot_col(net_eval):
   return index of max element in net_eval; (float)

2) calc_ratio(main_matrix, pivot_col):
   return vector formed by elements of main_matrix.getCol(pivot_col)
   devided by elements of last column of main_matrix;
   (vector)

3) define_pivot_row(ratio): non-negative
   return index of min element of ratio; (float)

4) define_pivot_element(main_matrix, pivot_col, pivot_row):
   return main_matrix.getEl(pivot_row, pivot_col); (float)

5) define_basis(basis, pivot_row, pivot_col, func_coef);
   basis.setEl(pivot_row, func_coef.getEl(pivot_col));
   return basis; (vector)

6) update_main_matrix(main_matrix, pivot_row, pivot_col, pivot_el):
   create new_matrix;
   new_pivot_row = main_matrix.getRow(pivot_row).divideBy(pivot_el);
   new_matrix.setRow(pivot_row, new_pivot_row);
   for i, where i is other rows:
   ~~new_row = main_matrix.getRow(i).divideBy(main_matrix.getEl(i, pivot_col)~~
   buf = new_matrix.getRow(pivot_row).multiplyBy(main_matrix.getEl(i, pivot_col));
   new_matrix.setRow(i, main_matrix.getRow(i).subtract(buf));
   return new_matrix; (matrix)

7) define_basis_el(basis, pivot_row, pivot_col):
   basis_el.setEl(pivot_row, pivot_col + 1);
   return basis_el; (vector)

8) calc_profit(main_matrix, basis):
   return ~~transpose(transpose(main_matrix).multiply(basis))~~;
   main_matrix.transpose().multiplyBy(basis); (vector)

9) calc_profit(net_eval, func_coef, profit):
   return (func_coef.subtract(profit)); (vector)

10) check_net_eval(net_eval):
   if all elements of net_eval ≤ 0, then return true, else return false; (bool)