ASSIGNMENT III

Subject : Linked-list implementation
Advisor : Res. Assist. Selim YILMAZ
Due Date : 01.01.2020 — 23:59:59

Genetic Algorithm

1 Introduction

Genetic Algorithm, or GA, is a well-known evolutionary-based heuristic search algorithm used in artificial intelligence (AI) and computing. GA is inspired by the notion of *survival of the fittest*, a theory driven from biological evolution.

GA makes use of a population which is composed of individuals, also called chromosome, to solve both constrained and unconstrained optimization problem. Each chromosome is composed of genes, the smallest unit containing a genetic value. Refer to Fig. 1 for illustration.

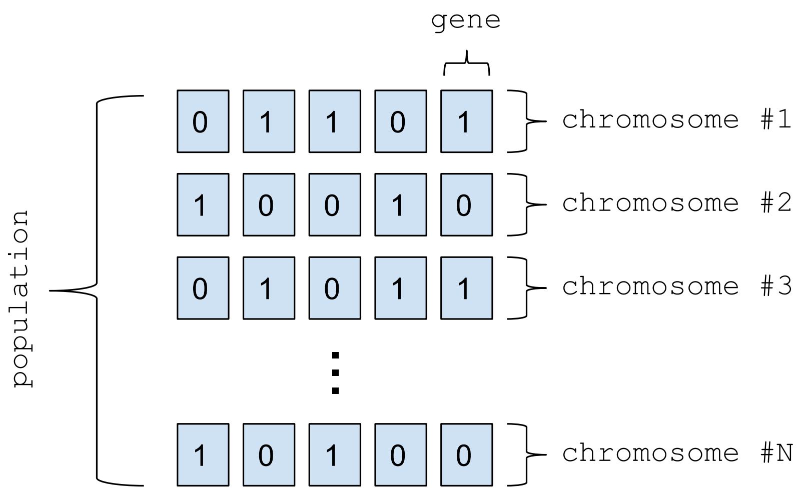


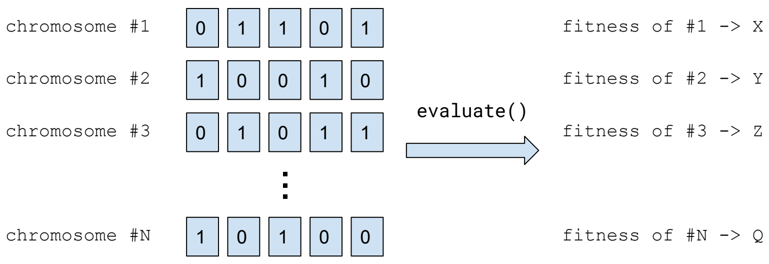
Figure 1: Demonstration of GA structure.

GA is an iterative algorithm and, at every step, it employs generic operators involving *selection*, *crossover*, and *mutation* to evolve new, hopefully better, individuals that represent the solution toward the problem at hand. The quality of each evolved chromosome is evaluated through a given fitness function. At the end of the evolutionary process, it returns the best solution found for the problem.

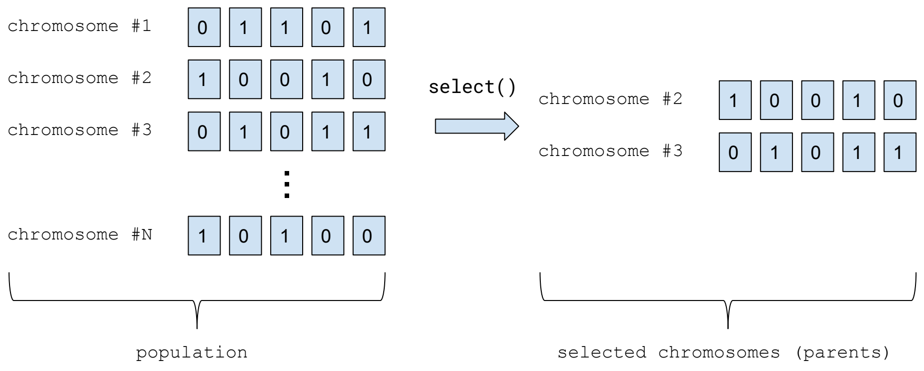
There are five steps in GA, which are briefly explained in order in the following:

- INITIALIZATION: simply creates a new population as illustrated in Fig. 1.

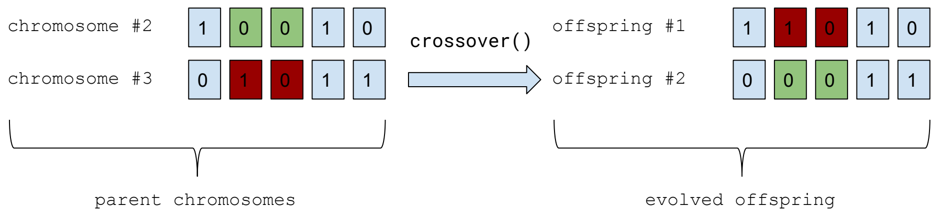
- FITNESS EVALUATION: reveals how well a chromosome represents a solution toward a problem.



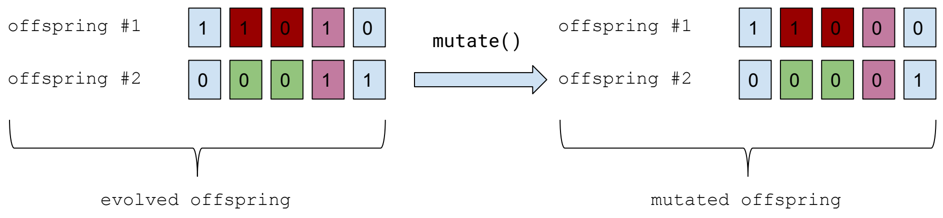
- SELECTION: selects two chromosomes as parent with respect to some rules:



- CROSSOVER: combines the selected parent to evolve offspring for next generation.



- MUTATION: applies a change on genes of evolved offspring.



The pseudo-code of GA is given in Algorithm 1

```
Initialize population;
Evaluate the fitness of each chromosome;
repeat
    Selection;
    Crossover;
    Mutation;
    Evaluate the fitness of evolved chromosome;
until termination criterion is satisfied;
return best-of-run chromosome;
```

Algorithm 1: The general steps of GA

2 The Task: Implementation of GA

In this assignment, you are expected to implement a basic version of GA. For the sake of clarity, some rules adopted in GA will be discarded here. To do that, you are expected to use **only** linked-lists as well as primitive types in `C` like int, char, and etc. so that you can get practice on it. In other words, arrays-like data structures will not be allowed to use.

In the following, every step that is vital to reach the objective of this assignment is explained:

i. *Program argument:* You should provide the following arguments to your implementation in order:

   - `Problem Size, PROB_SIZE`: This represents the size of the problem that is to be handled. So, the number of genes in any chromosome should be equal to `PROB_SIZE`.
   - `Population Size, POP_SIZE`: This represents total number of chromosomes in population.
   - `Generation Size, MAX_GEN`: This represents total number of iteration that GA needs to terminate evolutionary process.

ii. *Initialization of population:* As in GA, it should be the first step of your implementation. GA initially generates the chromosomes in a random manner, which results in different outcome every time it runs. Unlike to GA, your implementation should consider the content of a file named `population` to construct the initial population. Every line in this file represents a chromosome and every letter in any line separated with (:) represents the gene.

    In the case where the content of `population` is given as the left part of Fig. 2; population, chromosome, and the genes should be constructed as what given at the right part of the figure.

    As seen in the figure, the object `population` points only the head chromosome, i.e., chromosome #1.

    Every chromosome should point the next chromosome except the last chromosome, it is

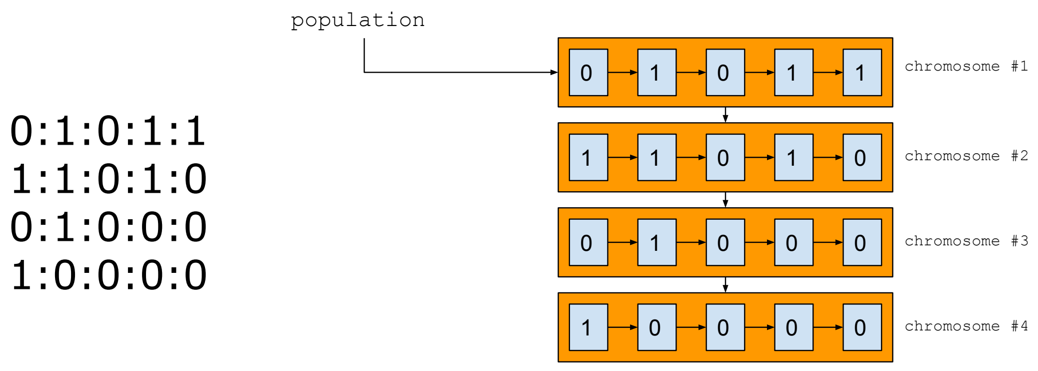


Figure 2: Generation of initial population w.r.t. `population`

4th chromosome in the figure which points NULL. In addition, the chromosome objects should carry *fitness* and *rank* that are evaluated at every generation, which is explained later.

Every gene object in the chromosome should have a variable indicating 0/1 and it should point the next gene as well except the last gene. It is worth pointing out here that the head gene should be pointed by the chromosome itself so that it can be reachable.

iii. *Fitness evaluation and ranking calculation:* After the creation of initial population, you should evaluate every chromosome to reveal how well it is able to identify the problem itself. There are lots of real-world problems that can easily be handled by GA, each of which has its own objective. Hence fitness evaluation differs with respect to the problem itself. In this assignment, our objective is to obtain a chromosome where its decimal equivalent is close to 0. Therefore, the fitness of a chromosome is its decimal equivalent (see Fig. 3).

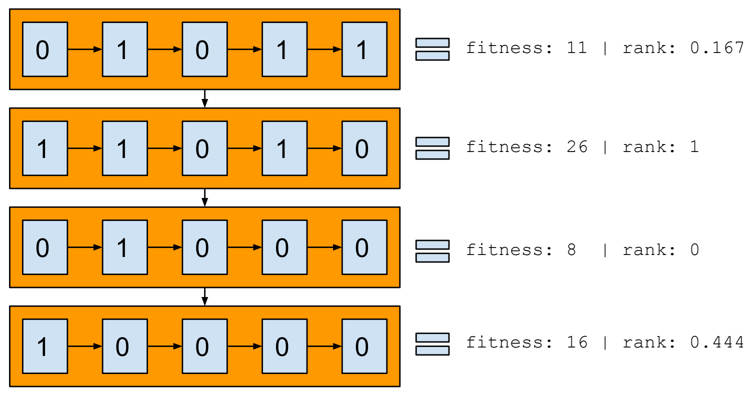


Figure 3: Fitness evaluation of initial population.

In ranking calculation, however, you need to calculate normalized values of a series of fitness values of all chromosomes, which hence should be in a range from 0 to 1 for best and worst chromosomes, respectively. The ranking calculation of ith chromosome is given in Eq. 1.

$$rank_i = \frac{fitness_i - fitness_{best}}{fitness_{worst} - fitness_{best}} \tag{1}$$

where $fitness_i$ is fitness value of ith chromosome and $fitness_{best}$ and $fitness_{worst}$ are the fitness values of best and worst chromosomes, respectively. Albeit rarely, there may be a case where $fitness_{best}$ is equal to $fitness_{worst}$, in such case $rank_i$ should be `NULL`.

iv. *Selection:* Unlike to GA that employs fitness proportionate selection, such as roulette wheel selection, tournament selection, and the like, in this assignment you are to consider a file, named `selection`, containing the IDs of chromosomes at every generation. An exemplar regarding the content of `selection` is given as follows:

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

here, total number of lines represents `MAX_GEN` in which `POP_SIZE/2` matches should be given. Every match can be separated by the white spaces. To elaborate further, 1st line of given file must be interpreted such that there are four matches among which 6th and 4th chromosomes should be selected to evolve their offspring, similarly 5th and 3rd chromosomes should be selected and so on at 1st generation.

v. *Crossover:* As explained in previous section, GA makes use of crossover to evolve new offspring from selected parents. Instead of generation of new chromosomes as in GA, you should perform crossover operation on parent chromosomes themselves, i.e., in-place crossover approach should be employed in this assignment. To do that, you are to consider a file named `xover` containing the beginning and the end indexes of genes that are going to be swapped. An exemplar regarding the content of `xover` is given as follows:

```
8:9
7:10
6:9
1:6
2:8
```

here, as in selection, total number of lines represents `MAX_GEN` and every line indicates the beginning and the end indexes of genes that are swapped. As can also be deduced from the file, index stated in this file can range from 1 to `PROB_SIZE`. For example, genes from 8th to 9th of every selected chromosomes should be swapped at 1st iteration. Figure 4 is given to demonstrate the 1st line of `selection` and `xover` files.

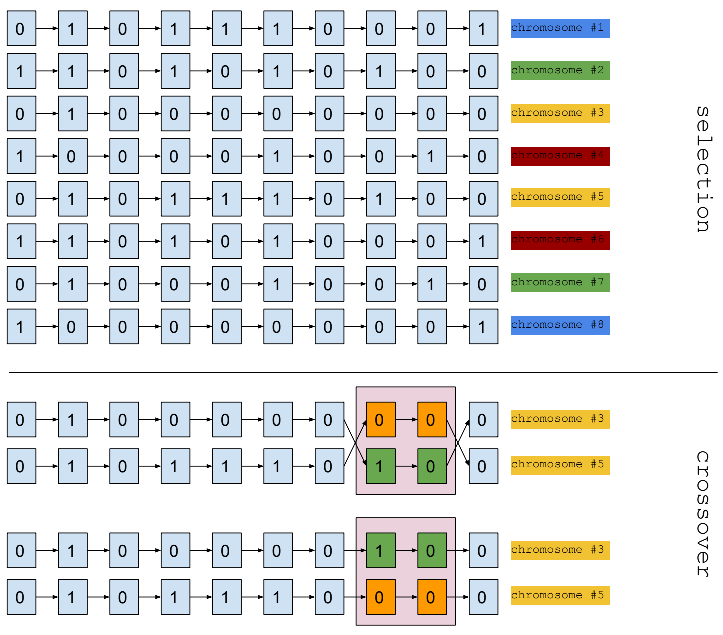


Figure 4: Selection and crossover operations.

vi. *Mutation:* As in GA, it mutates only one gene where the index is indicated in a file named `mutate`. An exemplar regarding to the content of `mutate` is given as follows:

```
5
9
9
3
7
```

here every line contains the index of gene that is going to be mutated just after the crossover operation. Through mutation, you should set the *complement* of the value at indicated value. For example, the first line of `mutate` shows that 5th gene of all selected chromosomes should be complemented and assigned (see Fig. 5).

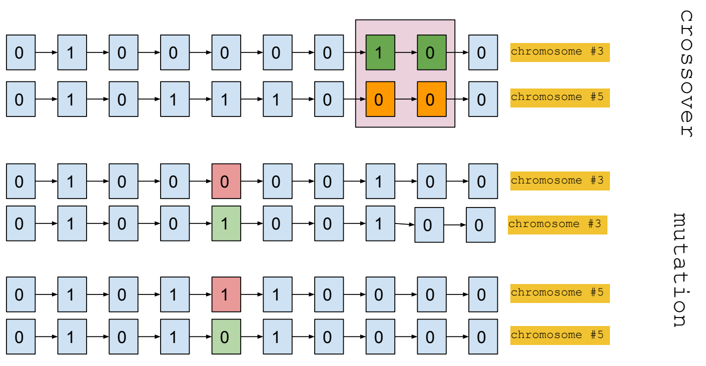


Figure 5: Mutation operations.

In addition to the operations explained above, your implementation should also have the following functionality:

- The best elite chromosome found until the iteration of interest should be stored to return it at the end of the evolutionary process.
- Chromosomes should be sorted with respect to the fitness value in ascending order and updated their positions in the list accordingly.
- Population should be printed to the screen with the fitness values followed by the best chromosome found so far.

The detailed version of **your GA implementation** is given as follows:

```
Read command line arguments;
Load all the parameters including selection, mutation, crossover;
Initialize population w.r.t. population file;
Evaluate the fitness and ranking of each chromosome w.r.t. Eq. 1;
Sort the population in ascending order;
Store the best chromosome;
Print the population with the best chromosome found so far;
repeat
    Apply selection;
    Apply crossover;
    Apply mutation;
    Evaluate the fitness and ranking of each chromosome w.r.t. Eq. 1;
    Sort the population in ascending order;
    Update the best chromosome if found better;
    Print the population with the best chromosome found so far;
until MAX_GEN is reached;
return best chromosome;
```

Algorithm 2: The detailed version of GA implemented in this assignment

An example:

Given that command-line argument is given as follows and the contents of `selection`, `xover`, `mutate`, and `population` files are as given thereafter:

```
./GA 10 8 10
```

population

```
0:1:0:1:1:1:1:1:0:1
1:1:0:1:0:0:1:0:0:0
0:1:0:0:0:0:1:0:0:1
1:0:0:0:0:0:0:1:0:1
0:1:0:0:0:1:1:1:1:0
0:1:0:1:0:0:0:0:1:1
0:0:0:0:1:0:1:1:1:1
0:1:0:0:1:0:0:1:1:1
```

selection

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

xover

```
8:9
7:10
6:9
1:6
2:8
5:8
3:5
1:7
4:7
1:5
```

Mutate

```
5
9
9
3
7
6
6
9
3
4
```

then the output should exactly match with the one given as follows:

```
GENERATION: 0
0:0:0:0:1:0:1:1:1:1 -> 47
0:1:0:0:0:0:1:0:0:1 -> 265
0:1:0:0:0:1:1:1:1:0 -> 286
0:1:0:0:1:0:0:1:1:1 -> 295
0:1:0:1:0:0:0:0:1:1 -> 323
0:1:0:1:1:1:1:1:0:1 -> 381
1:0:0:0:0:0:0:1:0:1 -> 517
1:1:0:1:0:0:1:0:0:0 -> 840
Best chromosome found so far: 0:0:0:0:1:0:1:1:1:1 -> 47
```

```
GENERATION: 1
0:0:0:0:0:0:1:0:0:1 -> 9
0:1:0:0:0:0:0:1:0:1 -> 261
0:1:0:0:1:0:1:1:0:1 -> 301
0:1:0:0:1:1:1:0:1:0 -> 314
0:1:0:1:0:1:1:1:1:1 -> 351
0:1:0:1:1:0:0:1:1:1 -> 359
1:0:0:0:1:0:0:0:0:1 -> 545
1:1:0:1:1:0:1:1:1:0 -> 878
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 2
0:0:0:0:0:0:1:1:1:1 -> 15
0:1:0:0:0:0:1:1:0:0 -> 268
0:1:0:0:1:0:1:0:1:1 -> 299
0:1:0:0:1:1:0:0:1:1 -> 307
0:1:0:1:0:1:0:1:0:1 -> 341
0:1:0:1:1:0:1:1:0:1 -> 365
1:0:0:0:1:0:1:0:0:0 -> 552
1:1:0:1:1:0:0:1:1:1 -> 871
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 3
0:0:0:0:0:1:0:0:0:1 -> 17
0:1:0:0:0:0:0:1:0:0 -> 260
0:1:0:0:1:0:1:0:1:1 -> 299
0:1:0:0:1:0:1:1:0:1 -> 301
0:1:0:1:0:0:1:1:1:1 -> 335
0:1:0:1:1:1:0:1:1:1 -> 375
1:0:0:0:1:0:1:0:0:0 -> 552
1:1:0:1:1:0:1:1:1:1 -> 879
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 4
0:0:1:0:0:1:0:1:0:0 -> 148
0:1:1:0:0:0:0:0:0:1 -> 385
0:1:1:0:1:0:1:0:0:0 -> 424
0:1:1:0:1:0:1:1:1:1 -> 431
0:1:1:1:0:0:1:0:1:1 -> 459
0:1:1:1:1:1:1:1:1:1 -> 511
1:0:1:0:1:0:1:1:0:1 -> 685
1:1:1:1:1:0:0:1:1:1 -> 999
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
```

```
GENERATION: 5
0:0:1:0:0:1:1:1:1:1 -> 159
0:0:1:0:1:0:0:1:0:0 -> 164
0:1:1:0:1:0:0:1:1:1 -> 423
0:1:1:1:0:0:0:0:1:1 -> 451
0:1:1:1:1:0:1:1:0:1 -> 493
0:1:1:1:1:1:0:1:0:0 -> 500
1:1:1:0:0:0:1:0:1:1 -> 907
1:1:1:0:1:0:0:0:0:1 -> 929
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 6
0:0:1:0:0:0:1:1:0:0 -> 140
0:0:1:0:1:1:0:1:1:1 -> 183
0:1:1:0:1:1:1:1:1:1 -> 447
0:1:1:1:0:1:1:0:1:1 -> 475
0:1:1:1:1:1:0:0:0:0 -> 496
0:1:1:1:1:1:0:1:0:1 -> 501
1:1:1:0:0:1:0:0:1:1 -> 915
1:1:1:0:1:0:0:1:0:1 -> 933
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 7
0:0:1:0:0:0:0:1:1:1 -> 135
0:0:1:1:1:1:1:1:0:0 -> 252
0:1:1:0:0:0:0:0:0:0 -> 384
0:1:1:0:1:0:1:1:1:1 -> 431
0:1:1:1:0:0:0:1:0:1 -> 453
0:1:1:1:1:0:1:0:1:1 -> 491
1:1:1:0:1:0:0:0:1:1 -> 931
1:1:1:0:1:1:0:1:0:1 -> 949
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 8
0:0:1:0:0:0:0:1:1:1 -> 135
0:0:1:1:1:1:1:1:1:1 -> 255
0:1:1:0:0:0:0:0:0:1 -> 385
0:1:1:0:1:0:1:0:0:1 -> 425
0:1:1:1:0:0:0:1:0:1 -> 453
0:1:1:1:1:0:1:1:0:1 -> 493
1:1:1:0:1:0:0:0:1:0 -> 930
1:1:1:0:1:1:0:1:1:0 -> 950
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
```

```
GENERATION: 9
0:0:0:0:1:0:1:1:1:1 -> 47
0:0:0:1:0:0:0:1:1:1 -> 71
0:1:0:0:0:0:0:1:0:1 -> 261
0:1:0:0:1:0:0:1:0:1 -> 293
0:1:0:0:1:1:0:0:0:1 -> 305
0:1:0:1:1:1:1:0:0:1 -> 377
1:1:0:0:0:0:0:1:1:0 -> 774
1:1:0:1:1:0:1:0:1:0 -> 874
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
GENERATION: 10
0:0:0:0:0:1:1:0:0:1 -> 25
0:0:0:1:1:1:0:0:0:1 -> 113
0:1:0:0:1:0:0:1:1:1 -> 295
0:1:0:1:0:0:0:1:1:0 -> 326
0:1:0:1:1:0:1:0:1:0 -> 362
0:1:0:1:1:0:1:1:1:1 -> 367
1:1:0:0:1:0:0:1:0:1 -> 805
1:1:0:1:0:0:0:1:0:1 -> 837
Best chromosome found so far: 0:0:0:0:0:0:1:0:0:1 -> 9
```

3 Grading Policy

Task	**Score**$^+$
Linked-list implementation⋆	**10**
Initialization of population	**5**
Fitness and rank evaluation	**5**
Selection	**10**
Crossover	**10**
Mutation	**10**
Sorting on population	**10**
Storing best individual	**5**
Printing in expected format	**5**
Adaptability to changing environment	**5**
Response time	**5**
Code readability	**5**
Report	**15**

* it is a must task without which your work is excluded from evaluation!

$$^+Final\ Score = Exec.\ Score + \left(Report\ Score \times \frac{Exec.\ Score}{85}\right)$$

Notes

- Do not miss the deadline.
- Save all your work **until the assignment is graded**.
- The assignment must be **original**, **individual work**. Duplicate or very similar assignments are both going to be considered as cheating.
- Regardless of the length, use **understandable** names to your variables, functions, and etc.
- Should you have a question, feel free to ask on Piazza. Be aware that the question has not been asked/discussed before.
- Do not submit your works via e-mail.
- You will submit your work from https://submit.cs.hacettepe.edu.tr/index.php with the file hierarchy as below:

This file hierarchy must be zipped before submission (Not .rar, only .zip files are supported by the system)

```
→ <student id>.zip
    → src.zip[1]
        → *.c
        → *.h
    → Makefile
    → report.pdf
```

[1]may contain subfolder(s)