CS300 – Spring 2020-2021 - Sabancı University

Homework #1 – Knight to the Rescue

Due 28 March 2021 22:00

Brief Description

In this homework, first, you will implement a program that will generate a chessboard of size NxN, where N represents the columns and rows. Then, you will need to implement a function that will place a certain number of pawns on the board based on given conditions. Finally, you will implement another function to place a knight and a king on the chessboard and check if the knight can rescue the king without touching any of the previously placed pawns.

To do this homework, you are required to implement a **Stack** using a **LinkedList** data structure, i.e., you can not use vectors in the stack implementation. The stack class **MUST** be template-based. You may want to store basic data types, structs or classes in this stack.

Program Flow

- **Chessboard Generation**

  The chessboard will be represented as an NxN 2-dimensional vector (i.e. matrix) where N is going to be an input to your program given by the user. The left bottom of the chessboard will be considered as the (1,1) coordinate. The size of the chessboard must be limited to be in this range $4 < N < 11$ where your program must do an input check.

  We expect each cell of the chessboard to be a **stackNode** that may contain the following information but you are not limited to storing only these fields, you may keep any other extra fields as needed:

  1. x_coordinate
  2. y_coordinate
  3. visited
  4. has_pawn

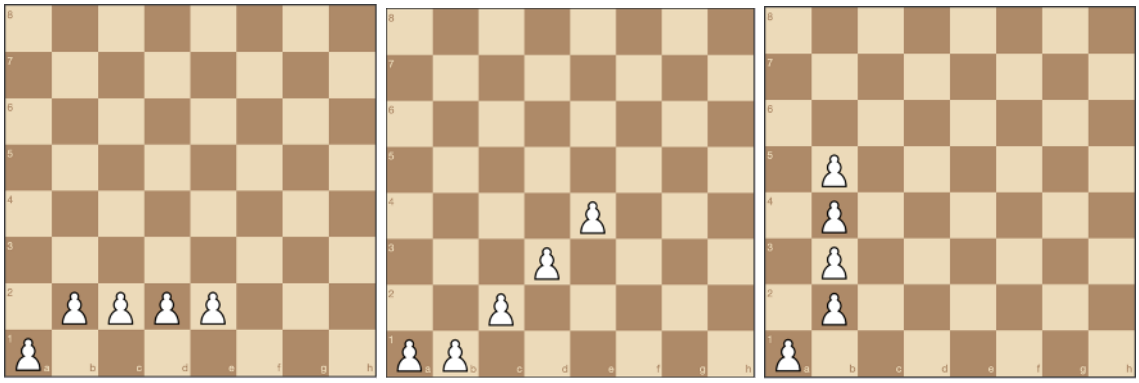
- **Pawn Placement**

  Once the chessboard is generated, you'll start to place the pawns with the following constraints:

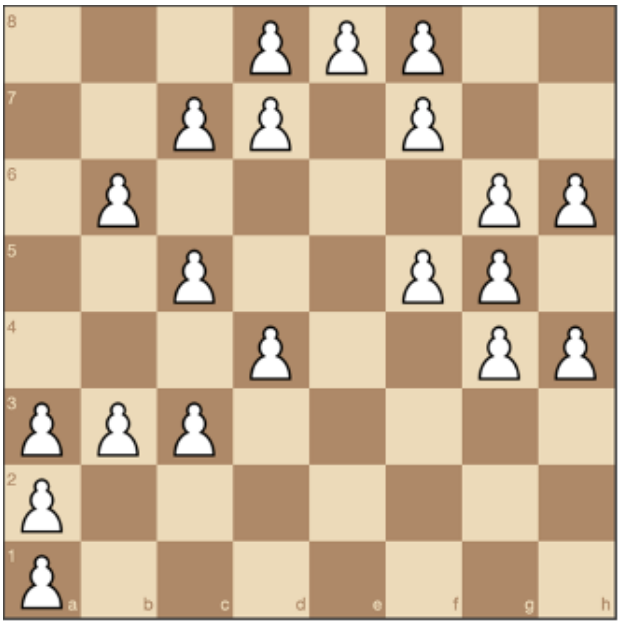
  - The number of pawns will also be an input to your program and must be limited to be 3N the most where your program must do an input check. For example, if N=8, the number of pawns can be 24 at most.
  - The first pawn will be placed at coordinate (1,1).
  - Each pawn following the previous pawn will be placed on a neighbouring cell depending on availability.
  - On the chessboard, each row, column and diagonal can hold a **maximum number of 3 pawns**.

Once all the pawns are all placed, your program will print out the coordinates of the pawns reading from your stack and also show the chessboard as a matrix.

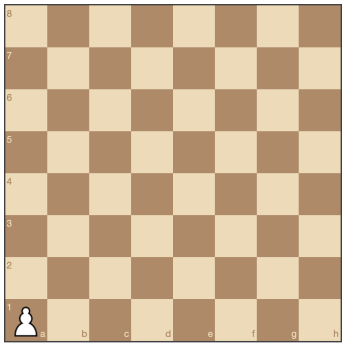
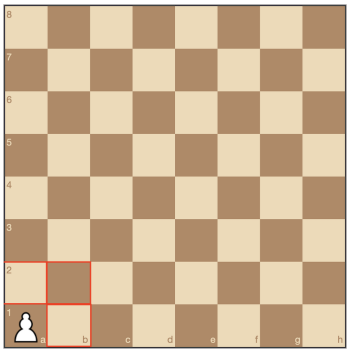
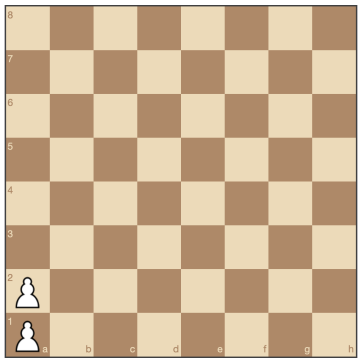
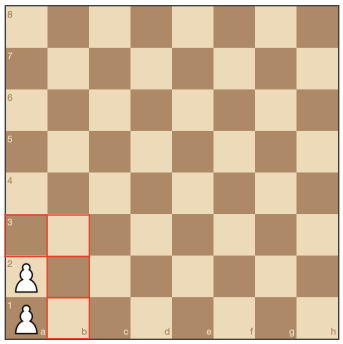
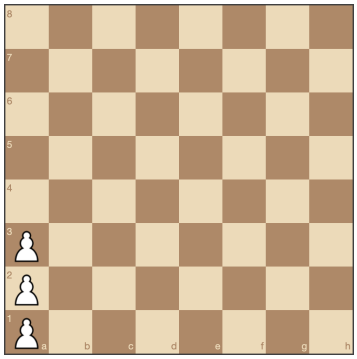
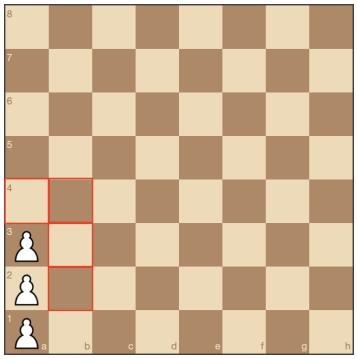
For example, the following placements of 5 pawns on an 8x8 chessboard are **invalid** because there are 4 pawns on a row in the first picture, on a diagonal on the second and a column on the third picture (your program must avoid such invalid placements):

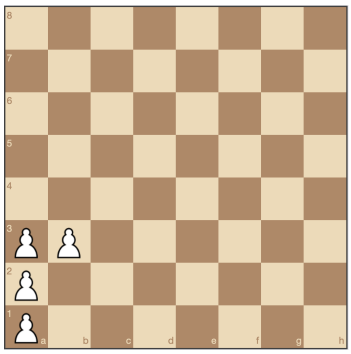
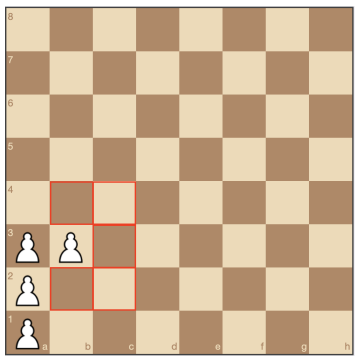
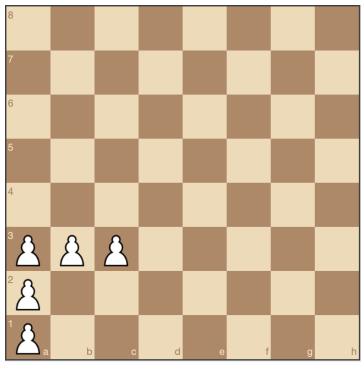
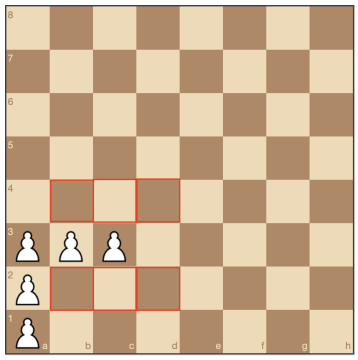
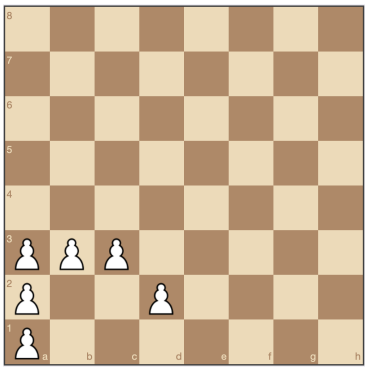
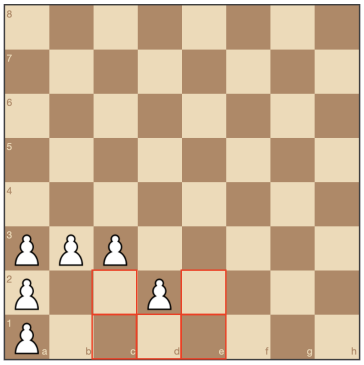
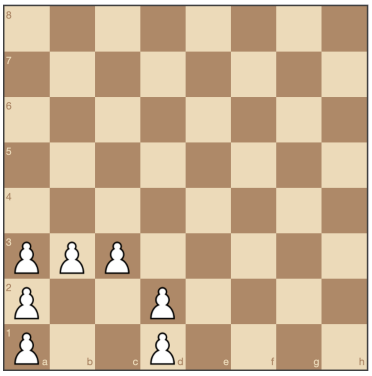
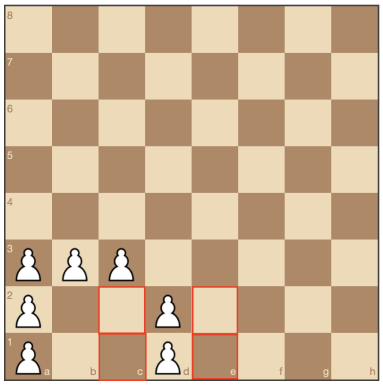


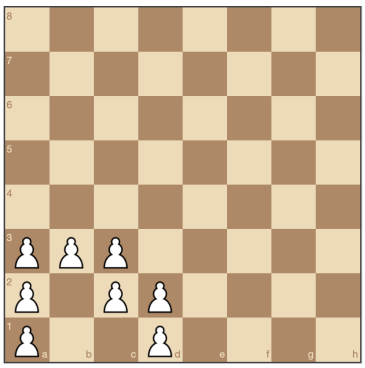
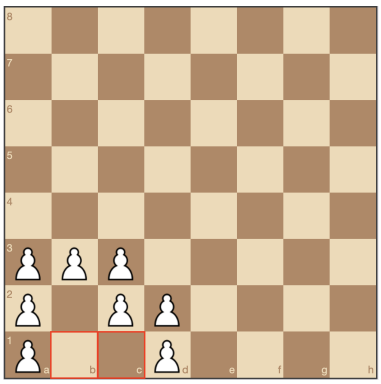
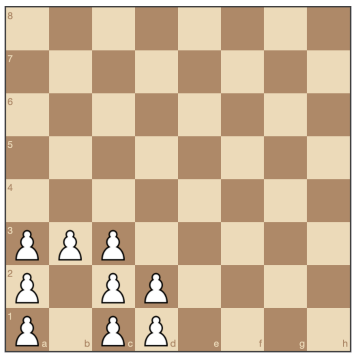
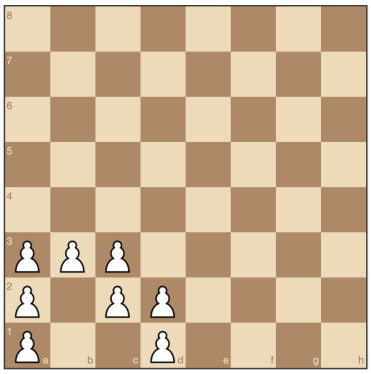
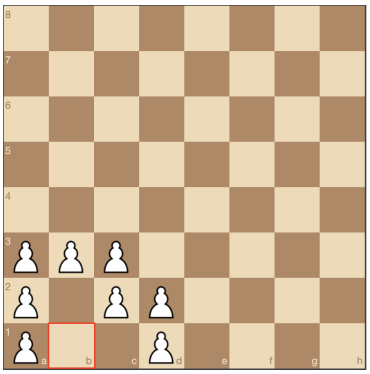
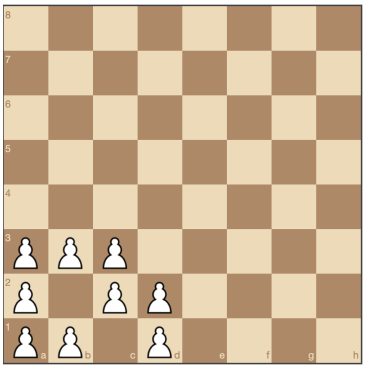
An example of a **valid** placement on an 8x8 chess boards with 20 pawns is below, note that there is no row or column or diagonal with more than 3 pawns in this chess board shown below:

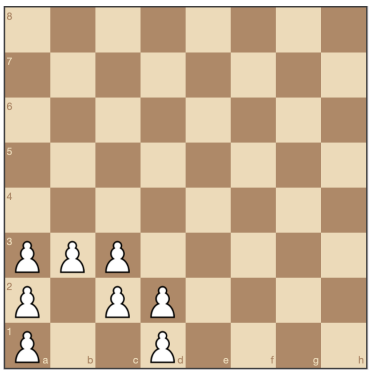
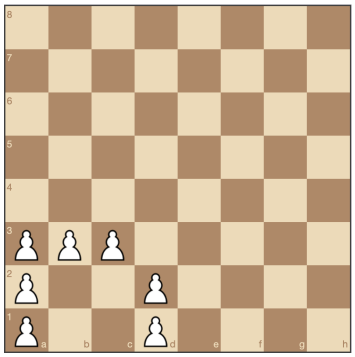
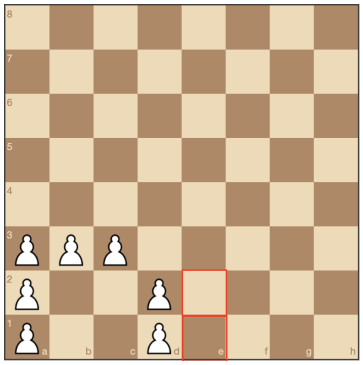


The following table shows the first 10 steps of possible pawn placements for a chessboard of size N=8 and p=20 many pawns. The possible cells that one can place each pawn in are shown as red bordered cells below; your program at each step will pick one of these red cells **randomly**.

Placement Trial	Chessboard	Available Spaces
1 Push (1,1)		
2 Push (1,2)		
3 Push (1,3)		

4 Push (2,3)		
5 Push (3,3)		
6 Push (2,4)		
7 Push (4,1)		

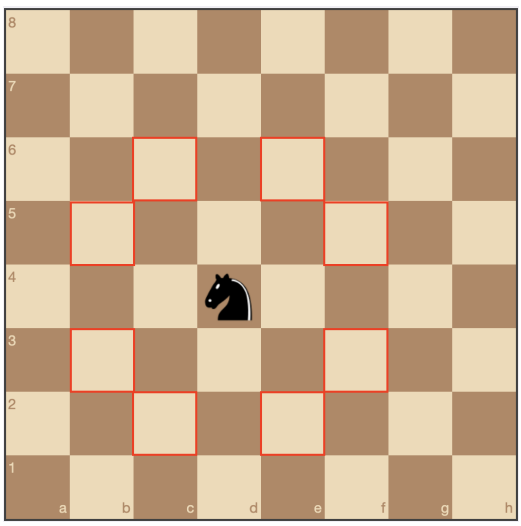
8 Push (3,2)		
9 Push (3,1)		No More Available Spaces But we still need to place more pawns
10 Pop (3,1) and mark unavailable		
11 Push (2,1)		No More Available Spaces But we still need to place more pawns

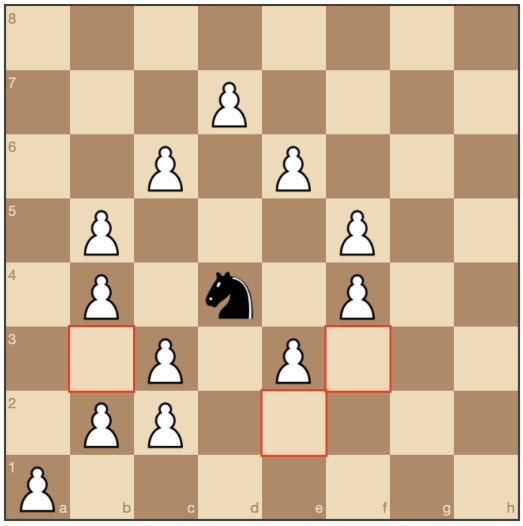
10 Pop (2,1) and mark unavailable		No More Available Spaces that we visited from last-placed pawn (3,2)
10 Pop (3,2) and mark unavailable		

- **Rescuing the King with the Knight**

  After the pawns are placed you'll place your knight and king on two given available cells on the chessboard where the user will input these cell coordinates as input to your program. Then, you'll check if your knight can rescue the king without touching any of the placed pawns on the chessboard. Then your program will print out the moves done by the knight. If there is no path to your king, then your program should indicate this by printing out **"No Way!"**.

  The knight moves in an "L-shape" — that is, they can **move** two squares in any direction vertically followed by one square horizontally, or two squares in any direction horizontally followed by one square vertically. **Your knight can not move to a cell that's occupied by the enemy pawn.** The following diagrams show the possible moves of a knight on the chessboard:





If your knight faces a deadlock where it can not move further and moves back (similar to the scenario that's explained for pawns above), this cell should be marked as unavailable. Also in your output, we shouldn't see moves like knight moves from (1,8) to (2,6) and back to (1,8). You should **display the path that the knight takes** to reach your king where the steps must not be repeated. That is, there shouldn't be any same coordinates in this path.

Output Format

The following example is the expected output format. You should write your output to a txt file named **output.txt**. Please note that the empty lines should be present in your output for it to work with the semi-automated grading tool. We expect **N, P, Pawn Locations, Knight and King locations, Knight Moves and the Chessboard matrix** (be careful to invert x and y values when you're printing the matrix out) in your output exactly as they appear below:

```
N = 8
P = 24

Pawn Locations
8 1
7 1
6 2
5 2
4 3
5 4
6 5
7 4
8 5
8 6
7 7
6 7
```

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

Knight = 1 8
King = 8 8

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

Matrix
0 0 1 1 1 0 0 0
0 0 1 0 0 1 1 0
0 0 1 1 0 0 0 1
0 1 0 0 0 1 0 1
0 1 0 0 1 0 1 0
1 1 0 1 0 0 0 0
1 0 0 0 1 1 0 0
1 0 0 0 0 0 1 1

Sample Runs

Sample Run 1

```
Please enter the size of the board (n): 8
Please enter the number of the pawns (p): 16
The matrix:
0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0
0 1 0 0 1 0 0 0
0 0 1 0 0 1 0 0
0 0 0 1 1 1 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
Enter the position of knight(x,y): 1 8
Enter the position of king(x,y): 8 8
No way!
```

Output File

```
N = 8
P = 16

Pawn Locations
6 4
5 4
6 5
5 6
5 7
4 8
4 7
3 7
2 6
3 5
4 4
3 3
2 3
1 3
1 2
1 1

Knight = 1 8
King = 8 8

No Way!
```

```
Matrix
0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0
0 1 0 0 1 0 0 0
0 0 1 0 0 1 0 0
0 0 0 1 1 1 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
```

Sample Run 2

```
Please enter the size of the board (n): 8
Please enter the number of the pawns (p): 14
0 1 0 0 1 1 0 0
0 1 1 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
Enter the position of knight(x,y): 1 8
Enter the position of king (x,y): 8 1
Path found, see the output file!
```

Output File

```
N = 8
P = 14

Pawn Locations
6 8
5 8
4 7
3 7
2 8
2 7
3 6
4 5
4 4
3 3
2 3
1 3
1 2
1 1
```

Knight = 1 8
King = 8 1

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

Matrix
0 1 0 0 1 1 0 0
0 1 1 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0

General Rules and Guidelines about Homeworks

The following rules and guidelines will apply to all homework unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS300. Office hours of TAs can be found here. Recitations will partially be dedicated to clarifying the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

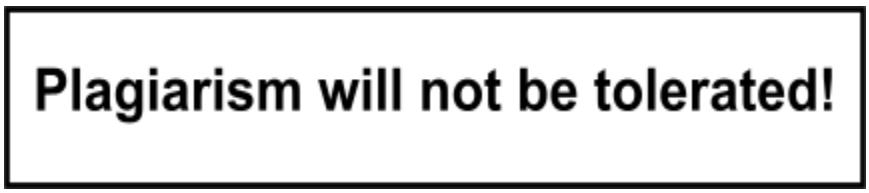
Please see the detailed instructions below/on the next page. The submission steps will get natural/easy for later homework.

Grading and Objections

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore, you should follow the guidelines about input and output order; moreover, you should also use the exact same prompts as given in the Sample Runs. Otherwise, the semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario, you will lose points.

Grading:

- Late penalty is 10% off the full grade and only one late day is allowed.
- **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long programs (which is bad) and unnecessary code duplications will also affect your grade.**
- Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/courses/cs201/plagiarism/



Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.

Grade objections: Since we will grade your homework with a demo session, there will be very likely no further objection to your grade once determined during the demo.

What and where to submit (IMPORTANT)

Submission guidelines are below. Most parts of the grading process are automatic. Students are expected to strictly follow these guidelines to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

- Use only English alphabet letters, digits or underscore in the file names. Do not use a blank, Turkish characters or any other special symbols or characters.
- Name your cpp file that contains your program as follows.

  "SUCourseUserName_yourLastname_yourName_HWnumber.cpp"
- Your SUCourse user name is your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse user name is cago, name is Çağlayan, and last name is Özbugsızkodyazaroğlu, then the file name must be:

  cago_ozbugsizkodyazaroglu_caglayan_hw1.cpp
- Do not add any other character or phrase to the file name.
- Make sure that this file is the latest version of your homework program.
- You need to submit ALL .cpp and .h files including the data structure files in addition to your main.cpp in your project.

Submission:

Submit via SUCourse+ ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).

Successful submission is one of the requirements of the homework. If for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!
Ulaş Eraslan, Ahmet Yasin Akyıldız and Gülşen Demiröz