CS1010S — Programming Methodology
National University of Singapore

Make-Up Practical Examination

22 November 2013

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam**.
2. This practical exam consists of **three** questions and you are expected to answer all **three** of them. The time allowed for solving this quiz is **2 hours**.
3. The maximum score of this quiz is **30 marks**. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
5. You should use the template `practical-template.py` provided, but rename your file `practical-exam-<mat no>.py` where `<mat no>` is your matriculation number. Please double check to ensure that there is no mistake. You are to upload your answers to IVLE Make-Up Practical Exam folder at end of the exam and you should only upload one file. If you upload multiple files, we will choose one at random for grading. Marks will be deducted if you fail to follow these instructions.
6. Please note that it shall be your responsibility to ensure that the correct file is uploaded to IVLE at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Make-Up Practical Exam. You have been warned.
7. In your submission, you are expected to comment out all the extraneous `print` statements that you might have added to aid debugging. Your failure to follow these instructions could potential cause a lot of problems in the grading and you would be penalized for it.
8. Please note that while sample executions are given, it is not sufficient to write programs that simply satisfy the given examples. Your programs will be tested on other inputs and they should exhibit the required behaviours to get full credit.

GOOD LUCK!

Question 1 : Reverse Even [10 marks]

A. Write a function `odd_position_digits` that takes in a positive integer and returns an integer consisting of only the odd-position (counting from the front) digits. [5 marks]

Sample execution:

```
>>> odd_position_digits(123456789)
13579

>>> odd_position_digits(1234567890)
13579

>>> odd_position_digits(1122334455)
12345

>>> odd_position_digits(183654729)
13579

>>> odd_position_digits(987654321)
97531

>>> odd_position_digits(123)
13
```

B. Write a function `reverse_even` that takes in a positive integer and returns an integer with the digits at the even positions (counting from the front) reversed. [5 marks]

Sample execution:

```
>>> reverse_even(123456789)
183654729

>>> reverse_even(1234567890)
1038567492

>>> reverse_even(1122334455)
1524334251

>>> reverse_even(183654729)
123456789

>>> reverse_even(987654321)
927456381

>>> reverse_even(123)
123
```

Question 2 : Building Maintenance [10 marks]

You are a new employee of the building maintenance department at NUS and you have just been given one data file `timetable.csv` in CSV (comma-separated values) format that contains timetabling information about NUS modules for **one week**. Your boss wants you to do some analysis of the building occupancy for the university.

A. **[Warm Up]** You are to write a function `available_venues` that takes as input a file and returns the number of different venues (column 8, 'Venue', of the data files) that are used by the various modules. [3 marks]

You are provided with two smaller sets of `timetable.csv` files and the following is the expected results for the two sets of files. Note that your code will be tested with other data files with the same input format, so just because your code works with the files provided does not necessarily mean that your code is correct.

Sample execution:

```
>>> available_venues("timetable1.csv")
350

>>> available_venues("timetable2.csv")
184
```

B. Your boss would like to understand the utilization of the available space in the school based on the timetables. You are to write a function `venue_occupancy` that takes as input a file and returns the average venue occupancy for all the venues in the university, i.e. you should find the venue occupancy for each venue and then find the average over all venues. For venue occupancy, we consider only office hours, i.e. between 800 and 1700, for weekdays (Mondays to Fridays). The venue occupancy of a venue is the ratio of the number of hours it is occupied to the total available hours it can be booked. Note that venues can be booked up to the minute, i.e. a booking of 2300 to 2359 is possible. **Hint:** You should look at columns 4, 5 and 6 of the provided data file. Since you are working with floating point numbers in this problem, there are likely to be some floating point errors. As long as your answer is to within 3 decimal places of the actual answer, your answer would be considered to be correct. [7 marks]

Also, while optimality is a core goal, your function is expected to be able to run reasonably fast and it should finish the computation within 30 seconds for a "regular" PC.

Sample execution:

```
>>> venue_occupancy("timetable1.csv")
0.12965079365079363

>>> venue_occupancy("timetable2.csv")
0.12898550724637659
```

Hint: You probably want to check your work as you go along instead of writing all your code and trying to debug everything at once. Consider yourself warned.

Question 3 : Fitness Test [10 marks]

Singaporean men has to do an annual physical fitness test, called IPPT, consisting of a number of stations. In this problem, you will model this scenario with two classes `IPPT` and `Station`.

`IPPT` only supports two functions:

1. `add_station` will take a string and an integer as arguments and returns a new fitness station. The string is the name of the station and the number is the initial number of people who start off waiting to take a test at that station.

2. `left` will return the number of people who are still currently taking the IPPT, i.e. they are at some station.

The order that the stations are added is important. The people taking the physical fitness test need to complete all stations in the IPPT before they are retired from the system. For people who still have stations left, they will move to the next station in cyclical order.

`Station` also supports two functions:

1. `complete` will take an integer as argument. The argument is the number of people who will complete the station. If the supplied argument is larger than the number of people currently waiting at a station, only the people at the station will complete the station. When people complete a station, they either (i) leave the system if they finish all the stations in the IPPT; or (ii) proceed to the next station.

2. `count` will return the number of people who are waiting to take the test at the station.

This problem is not quite as simple as counting the number of people at each station since each person starts off at a different station and will also therefore finish at a different station. Another thing: once that the test starts, i.e. `complete` is called by the stations, you can assume that no more stations will be added to the test.

Sample execution:

```
>>> ippt2013 = IPPT()

>>> chin_up = ippt2013.add_station("chin up",10)
>>> sit_up = ippt2013.add_station("sit up",10)
>>> sbj = ippt2013.add_station("broad jump",10)

>>> ippt2013.left()
30
>>> chin_up.count()
10
>>> sit_up.count()
10
>>> sbj.count()
10

>>> chin_up.complete(5) # First chin_up
```

```
>>> ippt2013.left()
30
>>> chin_up.count()
5
>>> sit_up.count()
15
>>> sbj.count()
10

>>> sit_up.complete(10) # First sit up
>>> ippt2013.left()
30
>>> chin_up.count()
5
>>> sit_up.count()
5
>>> sbj.count()
20

>>> sbj.complete(10) # First sbj
>>> ippt2013.left()
30
>>> chin_up.count()
15
>>> sit_up.count()
5
>>> sbj.count()
10

>>> chin_up.complete(10) # Second chin_up
>>> ippt2013.left()
30
>>> chin_up.count()
5
>>> sit_up.count()
15
>>> sbj.count()
10

>>> sit_up.complete(10) # Second sit up
>>> ippt2013.left()
30
>>> chin_up.count()
5
>>> sit_up.count()
5
>>> sbj.count()
```

```
20

>>> sbj.complete(10) # Second sbj
>>> ippt2013.left()
30
>>> chin_up.count()
15
>>> sit_up.count()
5
>>> sbj.count()
10

>>> chin_up.complete(10) # Third chin_up
>>> ippt2013.left()
25
>>> chin_up.count()
5
>>> sit_up.count()
10
>>> sbj.count()
10

>>> sit_up.complete(10) # Third sit up
>>> ippt2013.left()
15
>>> chin_up.count()
5
>>> sit_up.count()
0
>>> sbj.count()
10

>>> sbj.complete(10) # Third sbj
>>> ippt2013.left()
5
>>> chin_up.count()
5
>>> sit_up.count()
0
>>> sbj.count()
0

>>> chin_up.complete(10) # Final chin up
>>> ippt2013.left()
0
>>> chin_up.count()
0
```

```
>>> sit_up.count()
0
>>> sbj.count()
0
```

You may refer to Figures 1 to 11 to help you visualise how the situation evolves. To help you identify each individual person, the people involved are labeled from 01 to 30.

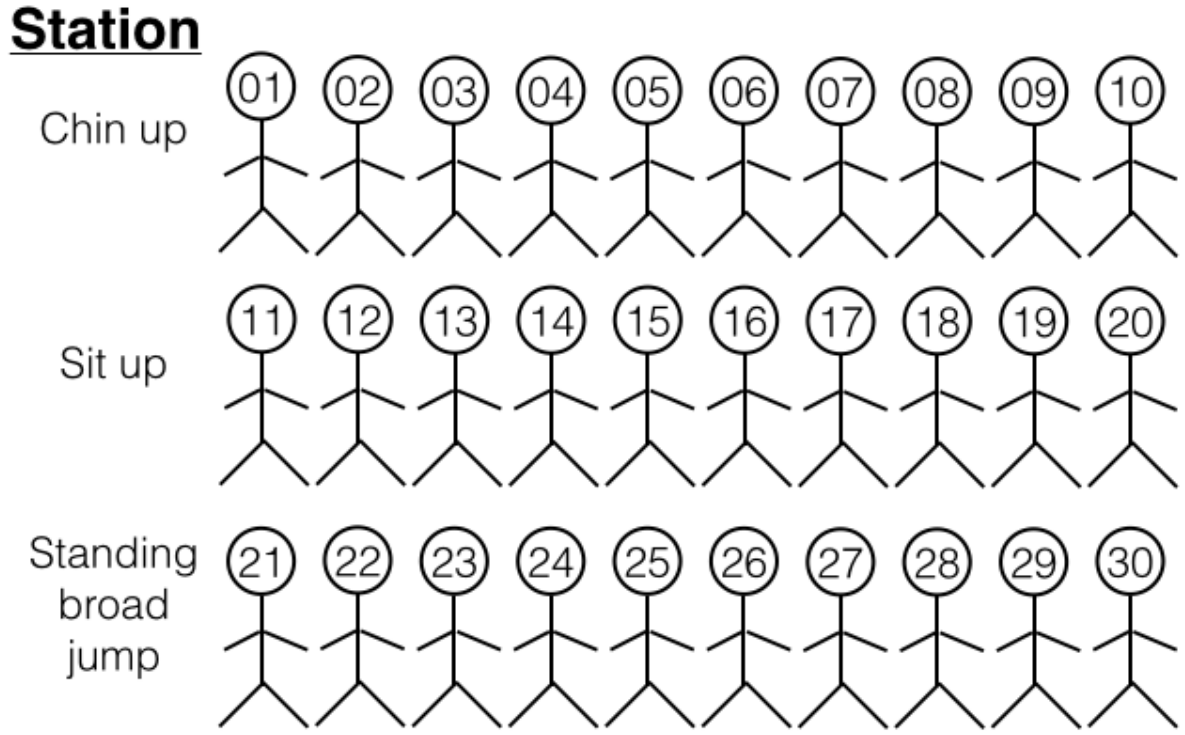

Figure 1: Initial configuration for *IPPT*.

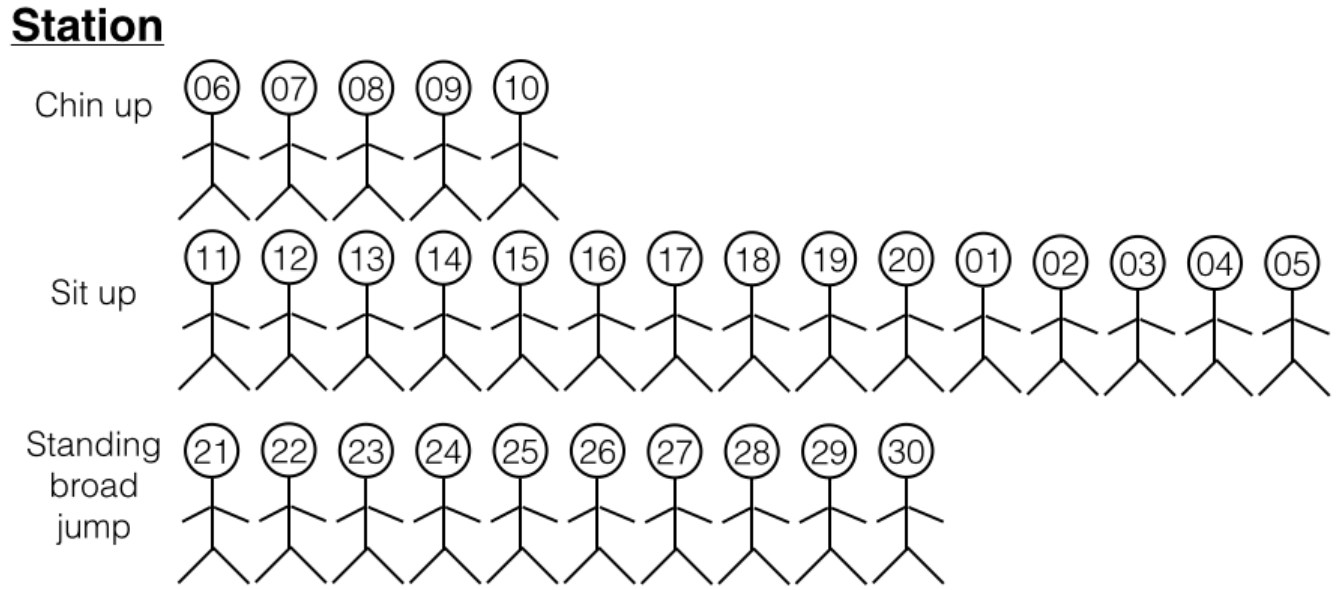

Figure 2: New configuration for Figure 1 after `chin_up.complete(5)`.

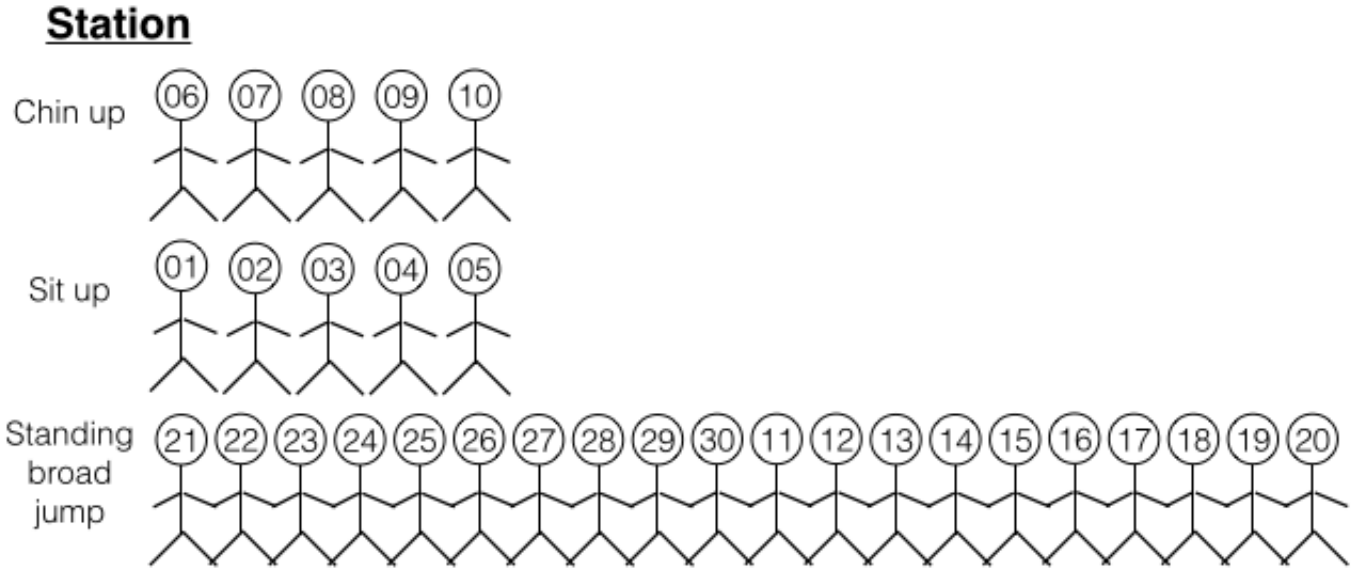

Figure 3: New configuration for Figure 2 after first `sit_up.complete(10)`.

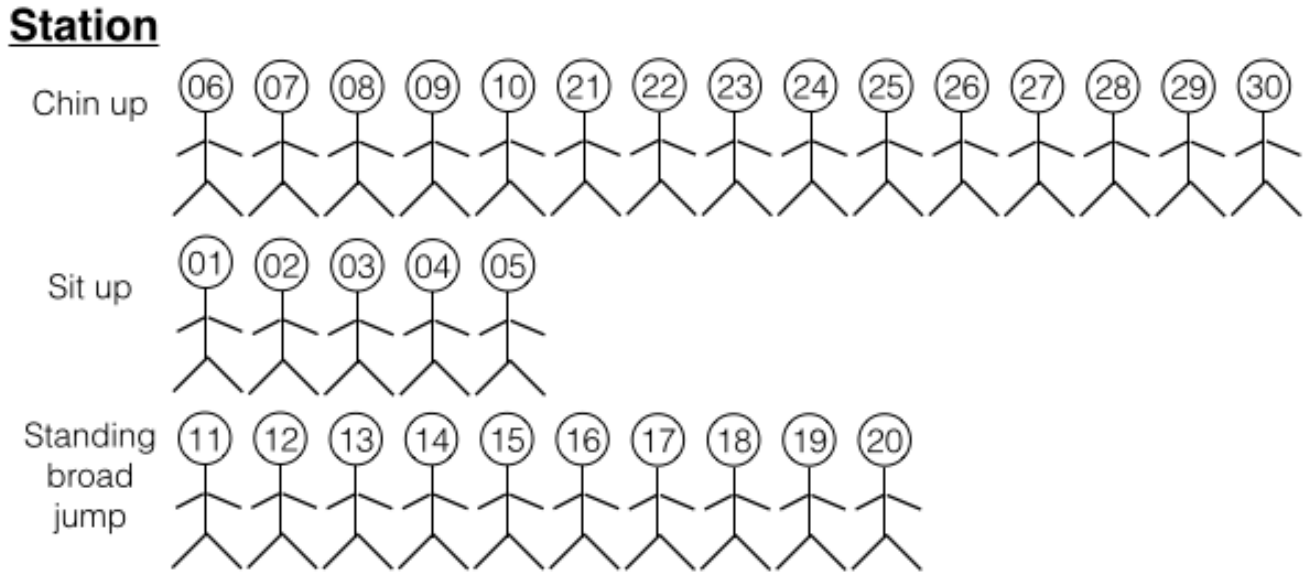

Figure 4: New configuration for Figure 3 after first `sbj.complete(10)`.

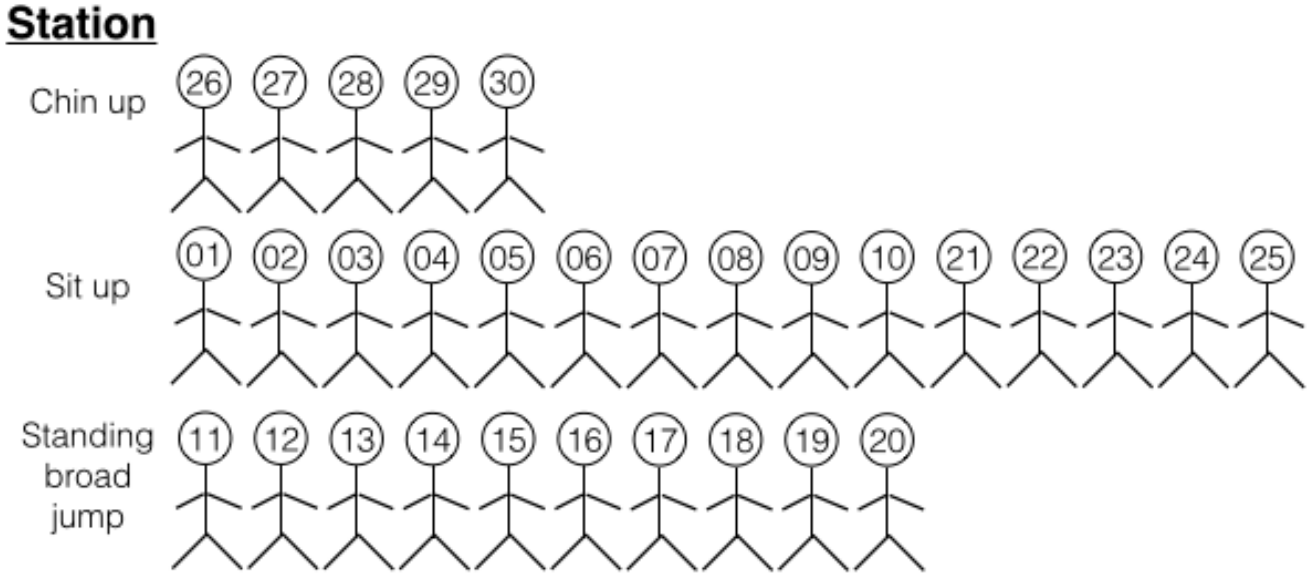

Figure 5: New configuration for Figure 4 after second `chin_up.complete(10)`.

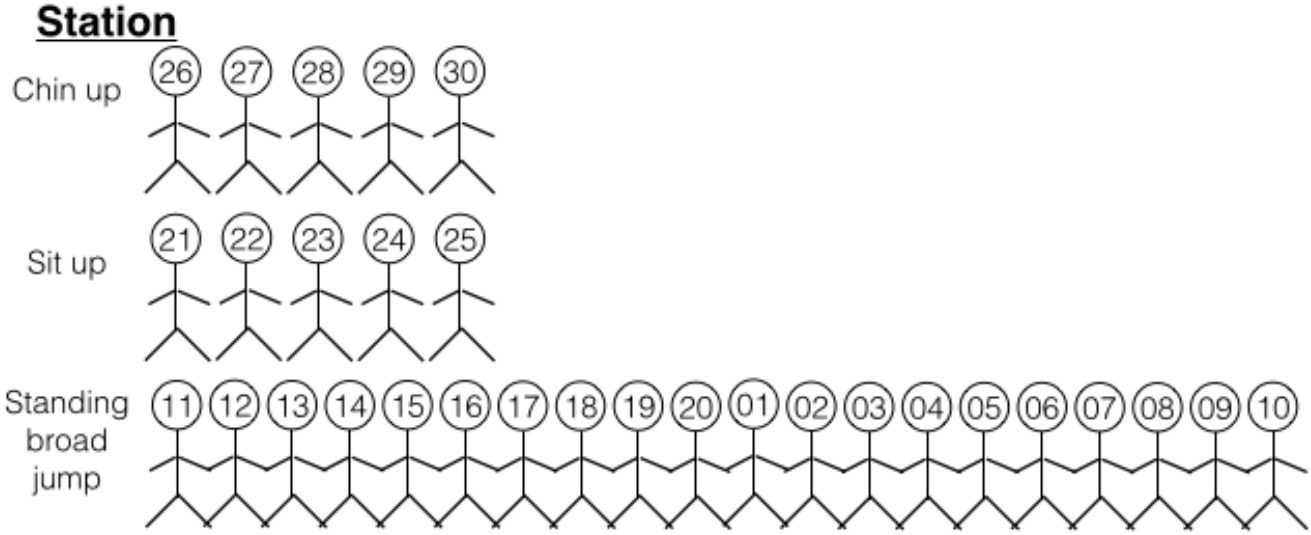

Figure 6: New configuration for Figure 5 after second `sit_up.complete(10)`.

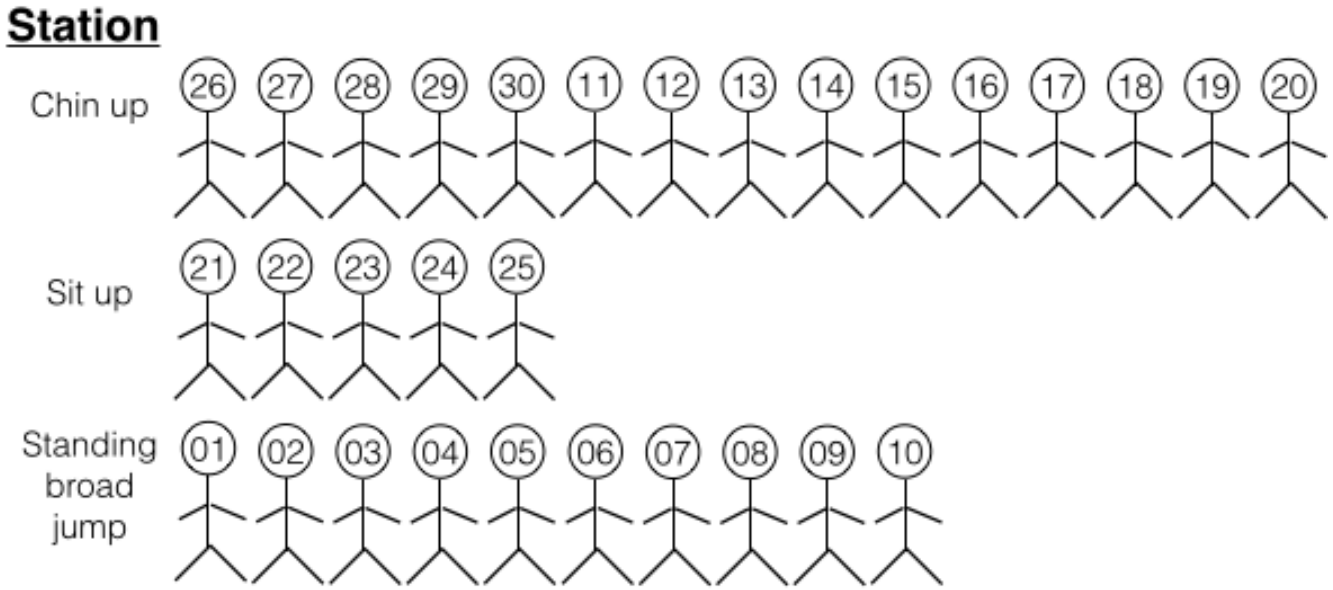

Figure 7: New configuration for Figure 6 after second `sbj.complete(10)`.

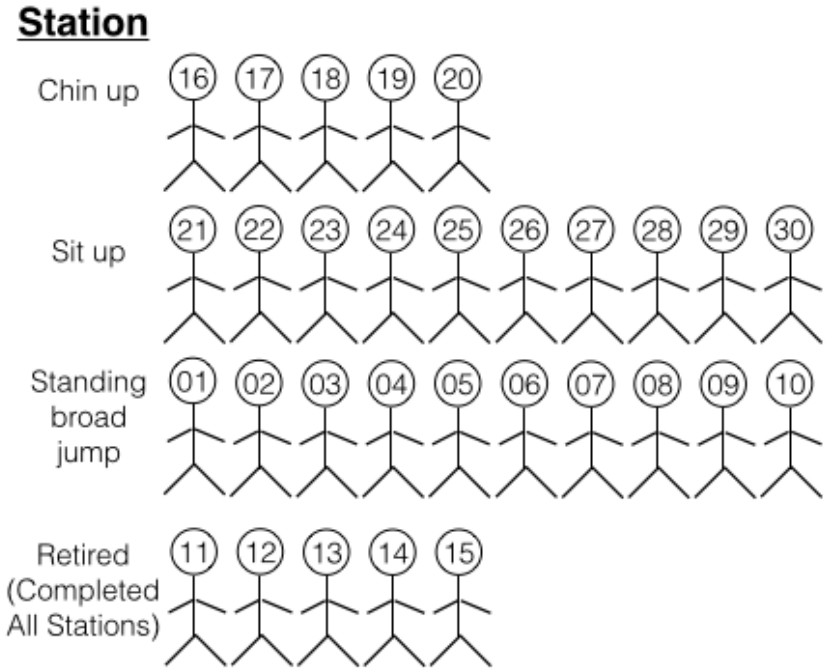

Figure 8: New configuration for Figure 7 after third `chin_up.complete(10)`.

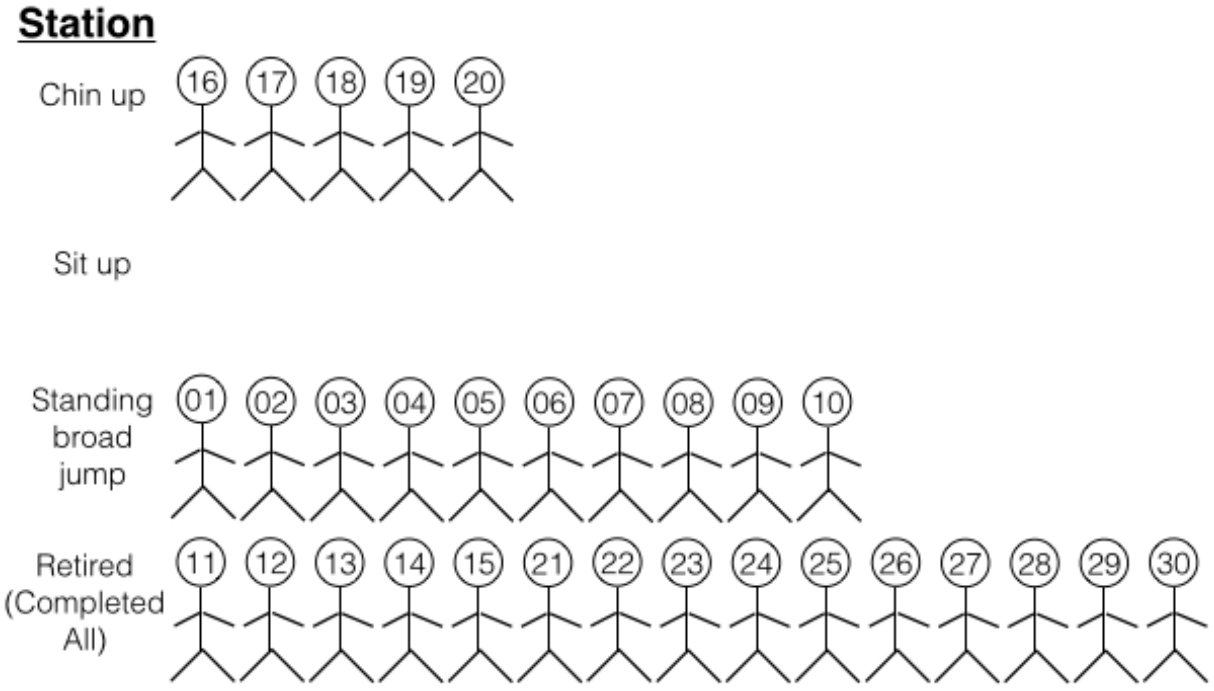

Figure 9: New configuration for Figure 8 after third `sit_up.complete(10)`.

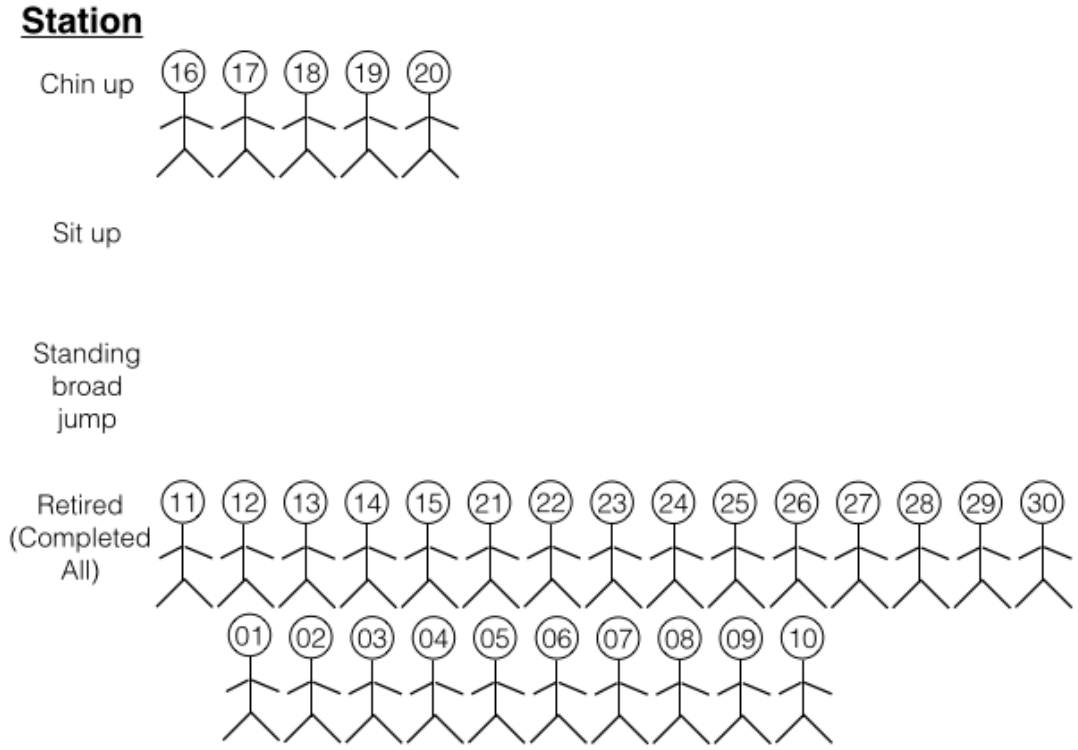

Figure 10: New configuration for Figure 9 after third `sbj.complete(10)`.

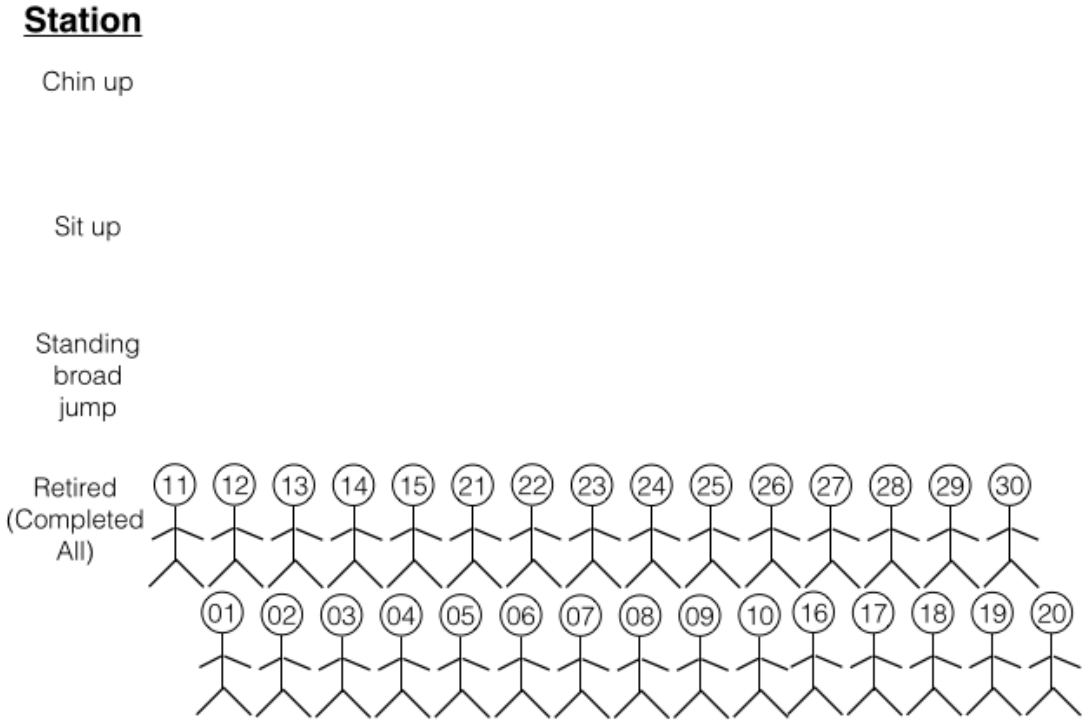

Figure 11: New configuration for Figure 10 after final `chin_up.complete(10)`.