1. Favourite Flavours

Standard Bank

Introduction

Isaac's Ice Cream wants to know which of its flavours are the most popular.

Task

Given the listing of sales for a particular day, output the flavour which had the most sales that day.

Example

On Tueday, Isaac's Ice Cream sold the following:

- one chocolate ice cream,
- one grapefruit sorbet,
- one mango ice cream,
- another chocolate ice cream,
- another mango ice cream, and
- another chocolate ice cream.

So, the most popular flavour on Tuesday was chocolate ice cream (with three sales).

Input

The first line contains a single integer D, the number of days.

The input data for each day starts with a line containing a single integer N, the number of ice creams sold on that day. This is followed by N lines, each containing a flavour, which is a single word of lowercase letters (`a-z`).

Sample input

```
2
6
chocolate
grapefruit
mango
chocolate
mango
chocolate
4
cappuccino
rainbow
rainbow
fudge
```

Output

For each day, output a line of the form

```
Case #X: F
```

where X is the day number, starting from 1, and F is the most popular flavour on that day.

Sample output

```
Case #1: chocolate
Case #2: rainbow
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq D \leq 20$
- $1 \leq N \leq 50$
- on each day, there is a unique flavour with the maximum number of sales

2. Water Bottles

Standard Bank

Introduction

You and your friends are going on a hike, and you are preparing your supplies for the hike. Specifically, you are preparing the liquid refreshments. Each of your friends has given you the exact amount of water that they need. However, you only have a limited number of bottles.

Task

Find the maximum number of friends you can prepare water bottles for.

You will be given a list of R requests for water, and a list of B bottles. Each request may only be stored in a single bottle (no amount may be split between multiple bottles) and no two friends can share the same bottle.

It is possible that while you may have enough bottles, not all of your friends' requests can be carried out because some of the bottles are too small. You must find the maximum number of requests you can carry out with these constraints.

Example

Suppose you have three amounts that you have to put into bottles: 4 litres, 6 litres, and 2 litres. You have four bottles, with capacities of 4 litres, 1 litre, 3 litres, and 5 litres, respectively.

The 4-litre request can be stored in the first 4-litre bottle. The 6-litre request cannot be carried out as we do not have a bottle big enough. Finally, the 2-litre request may be stored into the 3-litre bottle. So, a maximum of two requests can be carried out.

Input

The first line of input contains the integer T, the number of test cases in the input. Each test case is described by three lines.

The first line of each test case contains two integers R and B. The following line contains R integers r_1 to r_R, the amounts of the requests. The final line of the test case contains B integers b_1 to b_B, the capacities of the bottles that you have.

Sample input

```
1
3 4
4 6 2
4 1 3 5
```

Output

For each test case, output a line of the form

```
Case #X: N
```

where X is the test case number, starting from 1, and N is the maximum number of requests that you can satisfy.

Sample output

```
Case #1: 2
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 20$
- $1 \leq D \leq 10\,000$
- $1 \leq B \leq 10\,000$
- $1 \leq r_i, b_j \leq 100\,000$

3a. Transport Costs

Standard Bank

Introduction

You have been asked to help with planning routes for the truck drivers of a local company. Each driver has a different area in which he must transport goods along roads that connect towns in the area. A truck will consume a different amount of fuel for travelling along each road, depending on the length and quality of the road. You need to determine the minimum amount of fuel that is required to complete each transport job.

Task

A single transport job covers an area with N towns connected by R roads. Each road connects one town to a different town and requires a certain number of litres of petrol to travel. A road can be used to travel in either direction between the two towns it connects and no two roads connect the same pair of towns. Roads never meet outside of towns: once you start travelling on a road you can only continue to the town at the other end of the road.

Given all of this information, you must find the minimum amount of petrol required to travel from the first town in the area to the last town in the area.

Example

Suppose you are given an area with 4 towns connected by the following roads, along which a truck will consume the given amount of petrol:

- a road from town 1 to town 2 which consumes 8 litres
- a road from town 4 to town 2 which consumes 3 litres
- a road from town 1 to town 3 which consumes 4 litres
- a road from town 3 to town 4 which consumes 6 litres
- a road from town 3 to town 2 which consumes 2 litres

This is illustrated in Figure 1. The route from town 1 to town 4 which requires the least petrol is to travel from town 1 to town 3, then to town 2 and finally to town 4. Note that even though it is possible to get from town 1 to town 4 using only two roads, this route which uses three roads requires only 9 litres of petrol, which is less than any other route.

Input

The first line of input contains the integer T, the number of test cases.

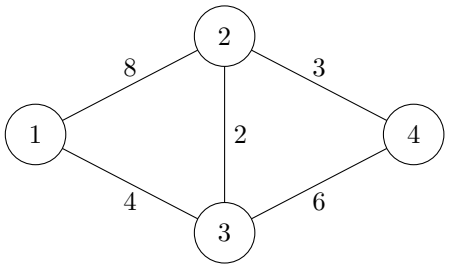


Figure 1

The input for each case starts with a line containing two integers, N, the number of towns in the area, and R, the number of roads, separated by a space. The following R lines each describe a single road. Each line consists of three integers separated by spaces: a_i and b_i, the towns connected by the road, and p_i, the number of litres of petrol consumed for using that road.

Sample input

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

Output

For each test case, output a line of the form

```
Case #X: P
```

where X is the test case number, starting from 1, and P is the minimum amount of petrol (in litres) to get from town 1 to town N in that case.

Sample output

```
Case #1: 9
```

Constraints

In the input used to test your program, the following constraints will hold:

- $1 \leq T \leq 20$
- $2 \leq N \leq 400$
- $1 \leq R \leq 20\,000$

3a. Transport Costs

- $1 \leq p_i \leq 100\,000$
- $1 \leq a_i, b_i \leq N$
- No road connects a town to itself ($a_i \neq b_i$ for all i)
- No two roads connect the same pair of towns
- It will always be possible to reach any town from any other town

4. Cryptomaton

Standard Bank

Introduction

Archeologists have discovered an ancient text-processing machine called the Cryptomaton. The museum which bought the machine is repairing it to working condition, and wants to ensure that the repairs do not change the behaviour of the machine.

Fortunately, a manuscript found with the machine has been decoded, and was found to contain a description of how the machine works.

Task

The Cryptomaton works by using R simple rules to expand the input message S to an enormous size. Each rule matches all occurrences of a character and replaces them with a specified output string. Characters that are not matched are left as they are. The output is repeatedly fed back into the machine in order to expand the text even further. In total, the machine is run G times.

These expanded strings can be very large and may require too much memory even on current computers. Therefore, you are asked to compute only the character at position P of the final string; this will be compared to the output of the Cryptomaton to check that it is working correctly.

Given S, G, P, and a set of replacement rules, compute the P^{th} character of the fully expanded string.

Example

Suppose the input string is 'ABC', $G = 3$ and $P = 5$, and the rules are:

- A $\rightarrow$ B
- B $\rightarrow$ C
- C $\rightarrow$ CA

The outputs of the three passes are:

- BCCA
- CCACAB
- CACABCABC

So the expanded string is 'CACABCABC', and the fifth character of this expanded string is 'B'.

Input

The first line of input is a single integer T, the number of test cases in the input.

The first line of each case contains the string S, and the second line contains the integers P, G and R. Each of the following R lines specify a single rule with two strings. The first string (the left-hand side of the rule) is a single character that should be replaced by the second string (the right-hand side).

Each string in the input test consists of uppercase letters (`A-Z`). Each rule has a distinct left-hand side.

Sample input

```
1
ABCDE
3 5 3
A B
B C
C CA
```

Output

For each test case, output a line of the form

```
Case #X: C
```

where X is the test case number, starting from 1, and C is the P^{th} letter of the expanded text.

Sample output

```
Case #1: B
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq$ length of S ≤ 100
- $1 \leq P \leq 1\,000\,000\,000$
- $1 \leq G \leq 30$
- $0 \leq R \leq 26$
- $1 \leq$ length of right-hand side ≤ 10

5. Electoral Districting

Introduction

After years of tyrannical dictatorship, the tiny country of Ruritania is becoming a democracy. The two political parties (which we will call "A" and "B") in Ruritania are dividing the country into D electoral districts in preparation for the first elections.

Each district will hold a separate election, and the candidate who recieves the most votes in a district becomes the member of parliament for that district. If the candidates from the two parties recieve the same number of votes in a district, then no member of parliament will be elected for that district.

You are part of a group of international observers monitoring the entire electoral process, and your team has been charged with ensuring the fairness of the districting process.

The results of the elections will depend heavily on how the country is divided into districts, so the politicians have agreed on a method of division which they claim is fair. You wish to prove to them that their method is not fair, so that they will choose a better method.

Task

Representatives from each party will take turns to create a district. One party draws a line from north to south or from west to east across the country. The part of country to the north or west of the line becomes a district, and the other party repeats the process with the remaining part of the country. This happens a total of $D-1$ times, until D districts have been created.

No district may be empty, so when creating a district, it must contain at least one voter, and it must be possible to divide the remaining part of the country into non-empty districts. No district boundary may be drawn through a voter's location.

Each party will conduct the district-drawing process in a manner which maximizes the difference between the number of districts they win and the number of districts won by the other party. They will do so optimally, and will assume that the other side is also doing so optimally.

Due to intensive pre-election polling, it can be assumed that everybody knows exactly who will be voting for each party.

Show that the method is unfair by calculating how the results are affected by which party goes first in the districting process.

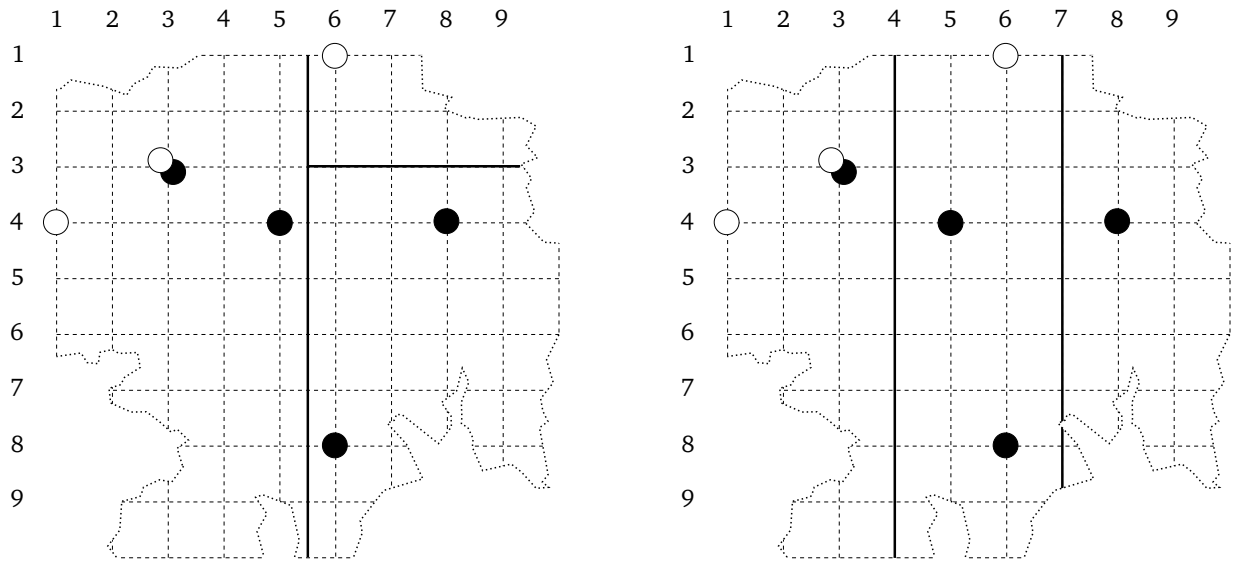
Example

Suppose there are only seven voters in Ruritania:

- supporters of party A at $(3,3), (8,4), (5,4)$ and $(6,8)$
- supporters of party B at $(1,4), (6,1)$ and $(3,3)$

and they are to be divided into three districts.

Note that is possible for voters, even supporters of opposing parties, to live in harmony at the same coordinates.



Example maps of Ruritania: possible districts if party A starts (left) and if party B starts (right).
Black dots represent party-A voters and white dots represent party-B voters.

5. Electoral Districting

Standard Bank

If party A starts, the districting process could proceed as follows:

1. Party A draws a north-south line at $x = 5.5$, creating a district to the west of that line where the two parties get two votes each.

2. Party B can then draw an east-west line at $y = 3$ so that the party-A supporter at $(6, 1)$ is in a district alone, and the remaining two party-B supporters form the remaining district.

In this case, the parties will win one district each with one district tied, so the margin between them is zero.

Surprisingly, it turns out that party A would do better by going second:

1. Party B draws a north-south line at $x = 4$, creating a district which it will win two votes to one.

2. Party A can then divide the remaining part of the country into two districts which it will win by drawing a north-south line at $x = 7$.

Here, party A will win two districts while party B will win only one, giving a margin of one.

In each case, there may be other lines which achieve the same result, but in no case can either party improve its margin by drawing a different line.

Input

The first line of input contains an integer T, the number of the test cases.

Each test cases consists of a line containing two integers, N (the number of voters in Ruritania) and D (the number of districts to be created). The next N lines each contain three integers, x, y and p, where (x, y) is the coordinate of a voter and p is the party they will vote for (either `A` or `B`).

Coordinates increase from north to south and west to east.

Note that although all voters are at integral coordinates, district lines can be drawn at any coordinates.

Sample input

```
1
7 3
1 4 B
3 3 A
8 4 A
6 1 B
5 4 A
6 8 A
3 3 B
```

Output

For each test case, output a line of the form

```
Case #X: A B
```

where

- X is the test case number, starting from 1;
- A is the number of districts won by party A minus the number of districts won by party B, if *party A* draws the first district;
- B is the number of districts won by party A minus the number of districts won by party B, if *party B* draws the first district.

Note that A and B could be negative.

Sample output

```
Case #1: 0 1
```

Constraints

It is guaranteed that in the input used to test your program:

- $1 \leq T \leq 10$
- $1 \leq D \leq 30$
- $1 \leq N \leq 10\,000$
- $1 \leq x, y \leq 20$
- It is possible to divide Ruritania into D non-empty districts

6. Yavalath on a Rhombus

Standard Bank

Introduction

Most games are invented by humans to be played by humans; some games are invented by humans to be played by computers. But recently, a few games are invented by computers: Cameron Browne's program LUDI tested out many different rule combinations, and evaluated how interesting the resulting games were. One of the games it invented was Yavalath, which is now published (to be played by humans) by *nestorgames*.

For the IT Challenge, we will slightly modify the board on which it is played, to make it easier for computers to play.

Task

The problem is an interactive task. You must write a program to play Yavalath against other programs.

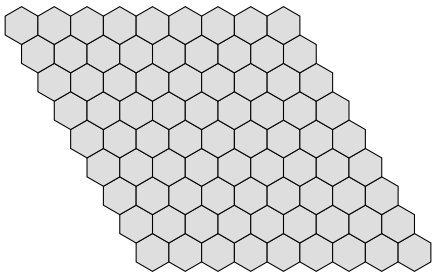


Figure 1: A Yavalath board.

Yavalath is a two-player game. Our version is played on an $N \times N$ rhombus* consisting of hexagonal tiles, as shown in Figure 1. Each hexagon is has a unique coordinate (R, C), designating the row and column respectively. The top-left tile is designated (1, 1), the top-right tile is (1, N), the bottom-left tile is (N, 1) and the bottom-right tile is (N, N).

The players take turns placing coloured tiles. The first player to place four (or more) pieces in a row wins. However, if a player makes a line of three pieces in a row without making a chain of four in a row at the same time, that player loses. If the board is filled without either player making a winning or losing move, the game ends in a tie.

Neighbouring Tiles

Given a tile at position (R, C) the following are its neighbours if they exist:

- West: $(R, C-1)$
- East: $(R, C+1)$

*The original Yavalath is played on a hexagon.

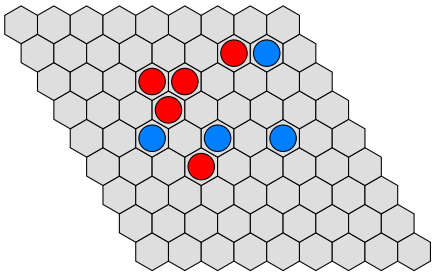


Figure 2: The blue player cannot prevent red from making four in a row without making three in a row herself.

- North-West: $(R-1, C)$
- North-East: $(R-1, C+1)$
- South-West: $(R+1, C-1)$
- South-East: $(R+1, C)$

Yavalath Tools

In the `yavalath_tools` directory on your USB drive are solution templates and a program for running games.

Templates

You will find a template program (in each language) on your USB drive, in the `templates` subdirectory of the `yavalath_tools` directory. This template includes all the code needed to communicate with the other player; you need only write code to decide which move you should make next.

You must complete the `move` method of the `YavalathPlayer` class. This method is called every turn with the position of your opponent's last move. The exception to this is at the start of the game where it is called with the position $(-1, -1)$. It must return the position of your next move. If you return an invalid move your program will be terminated.

You may also add initialisation code to the constructor, which is called at the beginning of each game with the current game's dimensions. You may add any other methods you need. Language-specific documentation for the template programs can be found on your USB drive at `yavalath_tools/templates/docs/index.html`.

Running games

The program called `RunYavalath` is a program for running games. The program can be run with either a graphical or command-line interface.

6. Yavalath on a Rhombus

The graphical interface can be run by double-clicking on the `RunYavalath` file and then selecting `Run` or `Run in Terminal` (the latter will allow you to see any debugging output your program produces). The GUI gives several options for each player:

- `Human` – this player will be controlled by the user
- `Random` – this player will play randomly
- `Random with fixed seed` – this player will play randomly, but if the same seed number is used multiple times, the same moves will be played, if possible (this may be useful for debugging)
- `Program` – your program will play as this player (for Java, select the `.class` file; for C++, select the executable file; for Python, select the source file)

  The following are hints for locations assuming that you used an Eclipse project, but you may need to adapt them to your project name and setup:

Java: `~/workspace/<project>/bin/Yavalath.class`
Python 2.x: `~/workspace/<project>/src/yavalath.py`
Python 3.x: `~/workspace/<project>/src/yavalath.py3`
C++: `~/workspace/<project>/Debug/yavalath`

Note that for Java and C++ programs, you might need to click `Build` in Eclipse to get the latest version of the file. For Python programs, it is essential that you use the correct extension: `.py` for Python 2.x, and `.py3` for Python 3.x. Using the wrong extension may result in failures both in `RunYavalath` and the submission system.

Below the players' area you can specify the size of the board and whether or not to visualise (display) the game on-screen. Once the game is set up, you can click `Play` to start it.

If `Human` is selected for either player, the game will be visualised regardless of the `Visualise` setting, to allow the user to play. If `Human` is selected for both players, two windows will be shown: one for each player. The visualisation adds a delay of about half a second per move; to time your program more accurately, disable the visualisation.

Constraints

When evaluating your submission:

- $N = 9$

Timing

Your program is allowed a total of 5 seconds of processing time during each game. Time taken by your opponent or the referee does not count against your program.

Scoring

There are two available points for this problem, as well as two possible time bonuses. The first point will be awarded to any program which consistently beats a random player. The remaining point and time bonuses will be awarded in a tournament.

When you submit a solution, it will play four games against a random player – two as the first and two as the second player. Your solution will be judged correct if it wins all the games.

A result of "wrong answer" means that your program lost at least one of the games. A result of "abnormal termination" means that your program exited (or crashed) before the game was over (unlike in non-interactive problems, exiting with an exit code of 0 before the end of the game is considered abnormal).

Once you have submitted a correct solution, you may continue to submit solutions, which will not change your score or total time. At the end of the contest, we will take the teams' last correct submissions and place them in a tournament to determine which solution is the best. The submissions will be ranked in terms of number of wins to give an overall score for the submission.

The top three teams in this tournament will receive bonuses: the first team will receive an extra point, the second team will receive a deduction of 120 minutes from their total elapsed time, and the third team will receive a deduction of 60 minutes from their total elapsed time.

If your program does not interact with the server validly for a given game, then you will automatically lose the game. A team must interact validly in at least one game to be eligible for a bonus. In the event of a tie, all teams tied for a position will receive the corresponding bonus.

Live tournament

During the contest, the judges will run games with accepted solutions from all teams, and show these games on a screen in the judging room. The results of these games will not be used in scoring. These games may not update immediately when a new submission is accepted.

The live games will not be run during the final hour of the competition.

The Standard Bank IT Challenge 2013

Competition rules

Rules for the Standard Bank IT Challenge 2013 are as follows.

The competition is sponsored by Standard Bank.

The Competition Chief Judge is solely responsible for ruling on any unforeseen situations and all such decisions are final.

1. Competition eligibility

1.1. To be eligible for the competition, students must
- currently attend a university in South Africa,
- be South African citizens or permanent residents,
- be full-time undergraduates in any discipline, or
- be full-time or part-time honours students in any discipline.

1.2 Up to four people can be entered for each team, all of whom may attend the heat or the final. In all teams one member must be named as Team Manager. All four members may take part in programming tasks. Each team must include at least one member of the opposite sex and at least one team member must be African, Asian, Indian or Coloured.

1.3 A team may not have more than two members of a team that came first, second or third in the finals of the competition in 2012.

1.4 For teams that successfully reach the finals, a maximum of two team members may be replaced by substitutes following the heats, as long as the conditions set out in (1.1), (1.2) and (1.3) are met.

1.5 Teams from Standard Bank may also participate but are not eligible for any of the prizes.

2. Conduct of the competition at the HEATS

2.1 The heats will be held on Saturday 20 April 2013 at the team's home university or nearby facility. The heats will last for 4 hours starting at 10:00 a.m.

2.2 Each team should be given access to one PC and one printer. The following programming languages may be used (participants can use any one or any combination of these)

a) Java 2 Standard Edition
b) C++
c) Python 2.7.3
d) Python 3.2.3

Solutions will be compiled and/or executed using Sun Java SE 1.6, GCC 4.4 and Python 2.7.3 and 3.2.3. If these versions or tools are changed, competitors will be notified before the start of the competition.

2.3 Only solution source files submitted via the contest submission system can be accepted or considered.

2.4 The Competition Managers cannot accept responsibility for any solutions that are not received during the heats, whether due to transmission delays or otherwise. Proof of transmission is not proof of receipt.

2.5 Up to five problems may be posed. There may be problems to which access will only be given once another problem has been solved.

2.6 As far as practicable, the problems will avoid dependence on detailed knowledge of a particular application area. The problems will not necessarily be of equal difficulty.

2.7 Competitors may bring into the contest and consult any source materials intended for human use such as for example:

- printed (hard copy) books and manuals
- printed (hard copy) program listings
- non-programmable calculators

Competitors may not load any material on the contest computers beforehand.
Competitors may not have access to the internet during the contest.
Competitors may not bring the following into the contest area, nor may they remove it from the contest area if issued there:

- computers, programmable calculators, personal digital assistants, personal music devices and similar
- mobile phones, tablets and similar
- machine readable media such as memory pens, CDs, etc
- computer peripherals

All source material must be declared. In the event of any uncertainty as to admissibility, the Chief Judge's decision is final. Infringement of the rule may lead to disqualification.

2.8 Any team which believes that a question is ambiguous or unclear may request clarification from the judges using the competition software. The judge's responses can subsequently be viewed using the competition software.

2.9 The competition rules may require problem solutions to conform to a specified interface or to produce output in a particular format. They will then be tested in a judged run against hidden test data compiled by the judges. By default, such a judged run is deemed successful by the judges if it processes the hidden test data correctly within the time and memory constraints that will be specified in the contest materials.

2.10 The scores of the other teams will be available on an electronic scoreboard. This scoreboard will not be updated during the last hour of the competition.

2.11 Problem solutions must not attempt to spawn new processes or threads, open files or sockets or otherwise attempt to interfere with the evaluation process. Teams submitting such solutions may be disqualified.

2.12 Teams will randomly be assigned to particular rooms, computing facilities, etc. While the judges will endeavour to minimise the effect of hardware failures, no responsibility can be accepted if they occur. Under no circumstances shall teams attempt to alter or interfere with the computing facilities provided without the prior permission of one of the competition judges.

2.13 Only the top-scoring team at each university is eligible to attend the final, and teams that do not correctly solve any of the questions are not eligible. The top nine eligible teams will be invited to the final. If there are fewer than nine eligible teams, the competition organisers may invite addition teams based on performance. This decision is entirely at the discretion of the competition organisers and no debate will be entered into in this regard

2.14 The decisions of the Competition Chief Judge are final and no correspondence will be entered into.

3. Conduct of the competition at the FINALS

3.1 The competition will be held on Wednesday 21 August 2013 in Gauteng.

3.2 No smoking will be allowed during the entire period of the competition. Contestants are allowed to apply nicotine patches or chew nicotine gum at their own risk.

3.3 Each team will be given access to one PC and access to a printer. The following programming languages may be used (participants may use any one or any combination of these)

a) Java 2 Standard Edition
b) C++
c) Python 2.7
d) Python 3.3

Solutions will be compiled and/or executed using Sun Java SE 1.6, GCC 4.4 and Python 2.7 and 3.3. If those versions or tools are changed, competitors will be notified before the start of the competition.

3.4 Teams will randomly be assigned to particular rooms, computing facilities, etc. While the judges will endeavour to minimise the effect of hardware failures, no responsibility can be accepted if they occur. Under no circumstances shall teams attempt to alter or interfere with the computing facilities provided without the prior permission of one of the competition judges

3.5 Up to seven problems may be posed. There may be problems to which access will only be given once another problem has been solved. As far as practicable, the problems will avoid dependence on detailed knowledge of a particular application area. The problems will not necessarily be of equal difficulty.

3.6 Competitors may bring into the contest and consult any source materials intended for human use such as for example

- printed (hard copy books and manuals
- printed (hard copy) program listings
- non-programmable calculators

Competitors may not load any material on the contest computers beforehand.
Competitors may not have access to the internet during the contest.
Competitors may not bring the following into the contest area, nor may they remove it from the contest area if issued there

- computers, programmable calculators, personal digital assistants, personal music devices and similar
- mobile phones, tablets and similar
- machine readable media such as memory pens, CDs, etc

- computer peripherals

All source material must be declared. In the event of any uncertainty as to admissibility, the Chief Judge's decision is final. Infringement of the rule may lead to disqualification or a time penalty at the discretion of the judges.

3.7 All problem solutions submitted for judging must be expressed in Java 2 Standard Edition, Python or C++ the designated languages of the competition, using the designated hardware with the designated operating system, compiler, interpreter and other software.

3.8 Teams may not accept help or advice on competition problems or rules from anyone except competition judges who are authorised to give such advice. The judges will also clarify problem ambiguities and may advise on system-related queries, explain error messages, etc.

3.9 The judges will not invite questions about problems. A team may submit a written assertion of ambiguity or error in a problem statement using procedures defined by the judges. If the judges agree that there is an ambiguity or error in the problem statement then they will issue a clarification to all teams.

3.10 The competition rules may require problem solutions to conform to a specified interface or to produce output in a particular format. They will then be tested in a judged run against hidden test data compiled by the judges. By default, such a judged run is deemed successful by the judges if it processes the hidden test data correctly within the time and memory constraints that will be specified in the contest materials.

3.11 Problem solutions must not attempt to spawn new processes or threads, open files or sockets, or otherwise attempt to interfere with the evaluation process. Teams submitting such solutions may be disqualified.

3.12 The solutions to some questions may be judged in "interactive mode". To be deemed correct, such solutions must conform to the specified interface and/or output format without causing exceptions or exceeding the time or memory constraints specified in the contest materials. Once a team has submitted a correct solution to an interactive question, they may submit further solutions to the same question without penalty. If more than one team submits a correct solution to such a question, the correct solutions will be ranked in a manner specified by the question. In general, this ranking will reflect the efficiency and/or fitness for purpose of the solutions. If a team has submitted more than one correct solution to the question, the last submitted solution will be used for the final ranking. At the discretion of the Chief Judge, the current ranking of solutions to an interactive question may be communicated to the competitors during the competition – otherwise, the judges' response to the submission of a solution to an interactive question will be in the same format as a response to a solution to any other question.

3.13 If a judged run is not deemed successful, then a statement to that effect will be returned by the judges. Any compiler error messages (e.g. identifying the place where an error was detected) will also be returned if they do not compromise the integrity of the judges' hidden test data. An unsuccessful judged run will be labelled with a phrase to indicate the reason for rejection, including for instance: syntax error, full-time error, time limit exceeded or incorrect output.

The judges will not intentionally attempt to mislead competitors about the nature of their errors; neither will they guarantee to identify the true error in the program. Normally, the first symptom of error will be noted and described by the most appropriate phrase.

3.14 Although the length of the competition is expected to be six hours, the judges have the authority to shorten or extend the competition in the event of unforeseen difficulties. If the duration of the competition is altered, every attempt will be made to notify competitors as soon as possible and to assure uniform impact on all teams.

3.15 Competitors will be informed of the results on the day of the competition. Again, the decision of the Competition Chief Judge is final.

4. Scoring of the competition in general

4.1 Sample data and sample output will be given with each question. This is provided to assist in clarifying the format required for input and output by the question, and should therefore be read as part of the question. It should be clearly understood that the only thing implied by the sample data is that a correct program, given the sample data, should produce the sample output. In particular it does not imply that the Judges will use that data to test the program, or that the sample data will necessarily check for any or all special cases that the question may require. Competitors can assume that the Judges' data will be in the correct format unless the question explicitly asks competitors to check the format.

4.2 The judges will be solely responsible for determining the correctness of submitted problem solutions and deciding the winners of the competition. They are empowered to adjust for and adjudicate on any unforeseen events and conditions. The decisions of the Chief Judge are final.

5. Scoring of the competition at the HEATS

5.1 The positions are based in the first instance on the number of successful judged runs. In the case of equal numbers of successful judged runs then the order is determined by the least amount of total elapsed time for all the successful runs.

5.2 A time penalty of twenty minutes will be added for every unsuccessful judged run on problems that are eventually completed successfully, with the exception of compilation errors for which there is no penalty.

6. Scoring of the competition at the FINALS

6.1 The positions are based in the first instance on the number of successful judged runs. In the case of equal numbers of successful judged runs then the order is determined by the least amount of total elapsed time for all the successful runs.

6.2 A time penalty of twenty minutes will be added for every unsuccessful judged run with the exception of compilation errors for which there is no penalty. Once a team has submitted a successful solution to an interactive question, subsequent unsuccessful submissions to the same question will not attract this penalty.

6.3 Each team which makes at least one successful submission to an interactive question is credited with a successful judged run for that question. At the discretion of the Chief Judge, these solutions may attract a score bonus. Typically, the highest-ranking solution will be credited with an additional successful judged run (i.e. will be given double weight in the final scoring of the competition). If there is a second-ranking solution, it will instead be credited with 120 minutes (i.e. 120 minutes will be deducted from the team's total elapsed time for scoring purposes) and the third ranked solution (if any) will be credited with 60 minutes. Any variation of this scoring for a particular interactive question will be communicated to the teams before the start of the competition.

6.4 Among teams solving the same number of problems during the competition, the number of minutes elapsed, including any time penalties, until each successful solution, will be totalled and these teams will be placed in order by the rule that the shorter the time taken, after any bonuses are taken into account, the higher the position.

6.5 The Judges reserve the right to add additional problems during the progress of the final. Should this occur, the scoring of such problems will be declared at the time.

6.6 If rules 6.1 to 6.4 still produce a tie, the tie will be broken by considering the time of the last correct submission from each team (teams with an earlier time rank higher than those with a later time). Only a team's first correct submission on any problem is considered in this rule (i.e. teams are not penalised for making further correct submissions on an interactive task). If a tie still remains, this rule will be repeated on successively earlier correct solutions until the tie is resolved.

7. Prizes

Prizes will be presented by the sponsor Standard Bank, to the university which comes in 1st, 2nd and 3rd place.

First	R170,000 to the university, the team represents
Second	R 20,000 to the university, the team represents
Third	R 10,000 to the university, the team represents

Prizes will be presented by the sponsor Dell, to the members of the team which comes in 1st and 2nd place.
Prizes will be presented by the sponsor DG Store, to the members of the team which comes in 3rd place.

8. General

8.1 The Competition Managers reserve the right to cancel the competition at any time, should it deem such action to be necessary.

8.2 The competition is not open to any person directly or indirectly involved in the management of this competition.

8.3 All entrants will be deemed to have accepted these rules and to agree to be bound by them when entering this competition.

8.4 The Competition Managers shall not pass on the personal details of any competitors to any third party without express consent except for the purposes of running the competition.

IT Challenge Finals contest manual

1. Introduction

For the finals of the IT Challenge, your submissions will be marked by Abacus, an automated judging system. However, you will not interact with Abacus directly. Instead, you will submit solutions on USB drives and the judges will load the solutions into Abacus.

2. Clarifications

If you are uncertain about some details in a task or the contest in general, you may submit a clarification request on the provided clarification request form. The team manager must take this form to the judges' room. Verbal requests for clarification about the tasks will not be accepted. Queries not related to the tasks (e.g., related to equipment problems) may be made verbally.

3. Writing your solution

If you are using Java, you can call your main class whatever you like, but it must be the *first* class in the file, and it must be `public`. You can include support classes after it, but they should not be `public`. Because the Java compiler requires source files to be named according to the package and class name, the marker has to extract these from the source file. To avoid the possibility of a compilation error, it is safest not to use the words `public`, `class` or `package` in comments before the declaration of the main class.

If you are using Python, you have a choice of Python 2.x or Python 3.x. Python 3.x is *not* backwards-compatible with Python 2.x. Therefore you *must* use the `.py3` extension for Python 3.x code and `.py` for Python 2.x code.

Your solution will read from standard input (`stdin`, `cin`, `System.in` or `sys.stdin` depending on language) and write to standard output (`stdout`, `cout`, `System.out` or `sys.stdout`). Your solution is tested by an automated marker, so you must be careful to input only what is asked for, and output only what is asked for. In particular, do not print prompts or messages like "`The answer is:`" unless you are asked to.

These instructions apply only to non-interactive questions. Interactive questions will contain separate instructions for class naming and I/O.

3.1. Floating-point output

In some questions, you may be asked to give output that is correct to a particular number of decimal places. It is recommended that you use the following code fragments to produce correct output for such questions (in all the cases below, we assume that the number of decimal places asked for is 3, and that the variable `x` is the value to be output):

- Python 2.x

```
print "Case #%d: %.3f" % (caseNum, x)
```

- Python 3.x

```
print("Case #{}: {:.3f}".format(caseNum, x))
```

- Java

```
System.out.println(String.format(
    "Case #%d: %.3f", caseNum, x));
```

- C++ with `stdio`

```
printf("Case #%d: %.3f\n", caseNum, x);
```

- C++ with `iostream`s

```
cout.precision(3);
cout << "Case #" << caseNum << ": "
     << fixed << x << endl;
```

4. Submitting

You will be provided with two USB drives. The drives are identical: you have two so that you can work with one while the judges have the other. Each drive will contain folders called `submit` and `marked`, each with a sub-folder per problem.

When you wish to make a submission, place a solution file in the per-problem sub-folder of `submit` (for example, if you have a solution called `prob3.cpp`, then copy it to `submit/3/prob3.cpp`). You must provide exactly *one* source file, since Abacus does not accept multi-file solutions, or multiple simultaneous submissions. Your solution must have the extension `.cpp`, `.java`, `.py` or `.py3`, as appropriate to the language it is written in. Please remember to safely eject the drive before unplugging it, as otherwise your solution may be missing or corrupted (if you are not sure how to do this, please ask for assistance).

You will also be given paper submission forms: use one per problem. Once you have placed your submission on the USB drive, the team manager must take the drive and the submission form for the problem to the judges' room. The judges will mark the solution, indicating the result of the submission on the submission form. They will move the solution to the `marked` folder on the drive before returning it.

The possible results of a submission are:

Compilation failure	Your submission did not compile. The judges' machines use the same compiler versions as the contestants', so this probably indicates that you did not correctly test that your solution compiles. The judges' can provide more details on the compilation error on request. There is no time penalty for compilation failures. Note that since Python is an interpreted language, there are no compilation errors — so be particularly careful before submitting, as what would be a compilation error in another language will be a run-time error in Python and will cost you penalty time.
Time limit exceeded	Your program exceeded the maximum time limit. This could mean that it is too slow or has an infinite loop.
Abnormal termination	Your program has crashed for some reason. Possible reasons include using too much memory, writing an excessive amount of output, throwing an uncaught exception, using illegal memory in C++, and returning a non-zero exit code (in particular, your `main` function in C++ must return 0).
Format error	Your program ran successfully, but did not conform to the specified output format. It might or might not have the right answers.
Wrong answer	Your program ran successfully, and at a glance produces the correct formatting, but the answers are wrong.
Correct answer	Well done!

If you wish to ask a question about a particular submission, you should submit a clarification request form and include the submission ID in your request.

5. Compiler versions

All submissions are evaluated on systems running Ubuntu 12.04, with the following compiler versions:

- GCC 4.6.3
- OpenJDK 6 (1.6.0)
- Python 2.7.3
- Python 3.2.3

6. Resource limits

Your program is subject to a number of limits on its resources:

Time	5 seconds. If you exceed this, you will get a response of “time limit exceeded.”
Memory	64MiB. If you exceed this, memory allocation functions will fail, usually leading to an abnormal termination.
Output	16MiB. If you exceed this, you will get an abnormal termination. Note that this is far larger than any correct output file, and is in place purely to protect the system from a program that outputs data in an infinite loop.

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Submission Record

Team:	**Problem:**

#	ID	Time	Result	Judge
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Standard Bank

Standard Bank IT Challenge 2013

Clarification Request

Team:	**Problem:**

Query:

Response:

☐ General announcement

Signed off by: _____ at _____

Standard Bank

Standard Bank IT Challenge 2013

Clarification Request

Team:	**Problem:**

Query:

Response:

☐ General announcement

Signed off by: _____ at _____

Standard Bank