This is the **solutions manual** for the **midterm contest**.

This problem set has **13 problems**. The items in the problem list below are links to each problem. You can click a problem name to jump to its page in the PDF

file.

Problem summary, in order of intended difficulty:

Problem
Level 1
Problem J: Convert
Problem F: Random Generator
Problem B: Countdown
Level 2
Problem E: User Accounts
Problem A: Range Sum Query
Problem C: Cancellable Quiz
Level 3
Problem L: Lower or Higher?
Problem H: Screen Scrolling
Problem I: Is it Sorted Again?
Level 4
Problem G: Paper Folding
Problem D: Printing Diamonds
Problem K: Information Tracker
Problem M: Conway's Game of Life

Solution for Problem J: Convert

For this problem, you just need to get the input, and apply the given formula.

Since the input can have decimals, you have to store the result in a `float` and not an `int`.

Solution:

```
Tc = float(input())
print( Tc*9/5 + 32 )
```

Solution for Problem F: Random Generator

First, we get the input. The input is a **list of integers**.

```
inputNums = [int(i) for i in input().split()]
```

After that, we can use the formula in the problem statement to get the output:

Solution:

```
inputNums = [int(i) for i in input().split()]

a = inputNums[0]
c = inputNums[1]
m = inputNums[2]
X_0 = inputNums[3]

print( (a*X_0 + c) % m )
```

Solution for Problem B: Countdown

First, we get the input. The input is **two integers on the same line**.

```
inputNums = [int(i) for i in input().split()]
```

We cannot use `inputNums = input()` only, because there is a space between the two numbers.

To make things easier, we can make variables for the inputs:

```
a = inputNums[0]
b = inputNums[1]
```

We want to print: $b, b-1, b-2, b-3, \ldots, a$.

This is a **range with start point b, end point $a-1$ and step -1.**

Solution:

```
inputNums = [int(i) for i in input().split()]

a = inputNums[0]
b = inputNums[1]

for x in range(b,a-1,-1):
    print(x)
```

Solution for Problem E: User Accounts

If you did not need to put extra zeros, the solution would be to use a loop:

```
inputNums = [int(i) for i in input().split()]

a = inputNums[0]
b = inputNums[1]

for x in range(a,b+1):
      print("team" + x)
```

However, the problem requires that we add an extra zero if the team number only has a single digit.
Note that if a team number has a single digit, it is **less than ten**.

Solution:

```
inputNums = [int(i) for i in input().split()]

a = inputNums[0]
b = inputNums[1]

for x in range(a,b+1):
      if x < 10:
            print("team0" + str(x))
      else:
            print("team" + str(x))
```

Solution for Problem A: Range Sum Query

First, we get the input:

```
N = int(input())
data = [int(i) for i in input().split() ]
endpoints = [int(i) for i in input().split() ]
A = endpoints[0]
B = endpoints[1]
```

We need to go through all the indices from A to B and add up all the elements of `data` in that range.

Solution:

```
N = int(input())
data = [int(i) for i in input().split() ]
endpoints = [int(i) for i in input().split() ]
A = endpoints[0]
B = endpoints[1]

sum = 0
for index in range(A,B+1):
      sum += data[index]

print( "RSQ("+str(A)+","+str(B)+")="+str(sum) )
```

Solution for Problem C: Cancellable Quiz

First, we get all the input:

```
N = int(input())
quizzes = [int(i) for i in input().split() ]
```

Now that we have the data stored in our program, we can think about how we can compute each of the required values.

- X is your friend's total score before his lowest quiz was cancelled.
  - We can calculate this by getting the ***sum*** of the elements of `quizzes`.
- A is the highest possible score before cancelling the lowest quiz.
  - Since there are N quizzes, and each quiz is worth 10 points, $A = N \times 10$.
- Y is your friend's total score after cancelling his lowest quiz.
  - We can calculate this by getting the ***least element*** in the `quizzes` array, then subtracting it from the ***sum of the elements*** in the quizzes array.
- B is the highest possible score after cancelling one quiz.
  - Since only one quiz gets cancelled, there are $N - 1$ quizzes remaining after cancelling one quiz. Since each quiz is worth 10 points, $B = (N - 1) \times 10$.

To get the sum of all elements in a list, we can start with 0, then add each of the elements of quizzes to it.

```
sum = 0
for x in quizzes:
      sum += x
```

To get the lowest of all elements in an array, we can start with the first element, and go through the other elements, **keeping track of the lowest element we've seen so far**.

```
minscore = quizzes[0]
for i in range(1,N):
      if quizzes[i] < minscore:
            minscore = quizzes[i]
```

Below is the full solution code.

Solution:

```
N = int(input())
quizzes = [int(i) for i in input().split() ]
sum = 0
for x in quizzes:
      sum += x
minscore = quizzes[0]
for i in range(1,N):
      if quizzes[i] < minscore:
            minscore = quizzes[i]
X = sum
A = N*10
Y = sum - minscore
B = (N-1)*10
print("Raw Score="+str(X)+"/"+str(A))
print("Final Score="+str(Y)+"/"+str(B))
```

Solution for Problem L: Lower or Higher?

First, we get the input. The first line of input tells us how many times we have to loop to get all the lines of input.

```
N = int(input())
for i in range(N):
      line = input()
```

Now let's consider how to handle one test case. Note that every test case contains:

`[integer] [word] [word] [word] [integer]`

so we can get the input for ever test case by **splitting the line, then converting the first and last words to integers.**

```
words = line.split()
A = int(words[0])
word1 = words[1]
word2 = words[2]
word3 = words[3]
B = int(words[4])
```

Note that the only things we need to check are

- Is `word2` `"higher"` or `"lower"`?
- Is $A < B$? Is $A > B$?

Here is the complete solution for problem L.

Solution:

```
N = int(input())
for i in range(N):
      line  =  input()
      words = line.split()

      A = int(words[0])
      word1 = words[1]
      word2 = words[2]
      word3 = words[3]
      B = int(words[4])

      if word2 == "higher":
            if A > B:
                  print("CORRECT")
            else:
                  print("WRONG")
      elif word2 == "lower":
            if A < B:
                  print("CORRECT")
            else:
                  print("WRONG")
```

Solution for Problem H: Screen Scrolling

First, note that we can ignore the grids. We only need to care about the array of integers which we are given as input. We have to "scroll" this array k elements to the right.

Let's call this array A.

Let's consider a case where we have the following array $(C = 10)$:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
0	1	2	3	4	5	6	7	8	9

Let's now consider the answer for several different values of k, maybe there's a pattern:

$k = 1$

9	0	1	2	3	4	5	6	7	8

$k = 2$

8	9	0	1	2	3	4	5	6	7

$k = 3$

7	8	9	0	1	2	3	4	5	6

We notice that when the array is "scrolled" k elements to the right, the result is something that looks like:

- The last k elements of the array
- Followed by the remaining elements of the array

How do we print **the last k elements of an array**?

Let N be the number of elements in an array.

We note that:

If $k = 1$, we need to print the last element, which has the index $C - 1$.
If $k = 2$, we print the elements at indices $\{C - 2, C - 1\}$.
If $k = 3$, we print the elements at indices $\{C - 3, C - 2, C - 1\}$.
If $k = 4$, we print the elements at indices $\{C - 4, C - 3, C - 2, C - 1\}$.
…

We notice the following pattern: to print the last k elements, we print the elements at all indices $C - k, C - k + 1, \cdots, C - 2, C - 1$.

```
for i in range(C-k,C):
    print( A[i] )
```

How do we print the **remaining elements of the array**?

Note that we have to print the elements we haven't printed yet.

This means all elements with indices from 0 to $C - k - 1$, because we already printed all elements from indices $C - k$ onward.

```
for i in range(0, C-k):
      print( A[i] )
```

Below is the full solution.

Solution:

```
inputNums = [int(i) for i in input().split()]
C = inputNums[0]
k = inputNums[1]
A = [int(i) for i in input().split()]

for i in range(C-k,C):
      print( A[i] )

for i in range(0, C-k):
      print( A[i] )
```

Solution for Problem I: Is it Sorted Again?

One method to solve this problem is to use the **All** pattern to check for the **three different conditions in the problem statement**.

```
increasing = True
decreasing = True
equal = True

for i in range(1,N):
     if not (A[i-1] <= A[i]):
          increasing = False
     if not (A[i-1] >= A[i]):
          decreasing = False
     if not (A[i-1] == A[i]):
          equal = False
```

Note that after the for loop, the variables will indicate if the sequence is increasing, decreasing or equal.

Then, we make sure we check each boolean in the proper order.

Since increasing and decreasing sequences are **not equal**, we have to be careful:

Solution:

```
N = int(input())
A = [int(i) for i in input().split()]

increasing = True
decreasing = True
equal = True

for i in range(1,N):
     if not (A[i-1] <= A[i]):
          increasing = False
     if not (A[i-1] >= A[i]):
          decreasing = False
     if not (A[i-1] == A[i]):
          equal = False

if equal:
     print("equal")
elif increasing:
     print("increasing")
elif decreasing:
     print("decreasing")
else:
     print("not sorted")
```

Solution for Problem G: Paper Folding

We can get a better idea of what to check by **labelling the grid with array indices**.

Here is a grid where $N = 5$.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Now, let's look at how the numbers will match up if we fold the grid of numbers.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Notice that a number at index (x, y) will be paired with a number at index (y, x).

Thus, the only thing we need to make sure is that for every index (x, y), the number in that position is the same as the number in the index (y, x).

This means we have to **store the input grid**, then **compare each pair of indices**.

Using the **All** pattern, we can check if the element at index (x, y) matches the element at index (y, x) for all indices.

Solution:

```
N = int(input())
grid = []
for row in range(N):
      inputRow = [int(i) for i in input().split()]
      grid.append( inputRow )

valid = True
for row in range(N):
      for col in range(N):
            if grid[row][col] != grid[col][row]:
                  valid = False

if valid:
      print("Yes")
else:
      print("No")
```

Solution for Problem D: Printing Diamonds

There are many ways to solve this problem. Here is one approach.

Here is a diamond where $N = 5$.

```
1      *
2     ***
3    *****
4   *******
5 *********
6   *******
7    *****
8     ***
9      *
```

We can consider printing the **top half** of the diamond first. Let's focus on the top half.

Let's denote spaces with periods, so that we have a better idea of how many spaces we need to print.

1	....*	For line 1, print 4 spaces then 1 asterisk.
2	...***	For line 2, print 3 spaces then 3 asterisks.
3	..*****	For line 3, print 2 spaces then 5 asterisks.
4	.*******	For line 4, print 1 space then 7 asterisks.
5	*********	For line 5, print 0 spaces then 9 asterisks.

Note that for line number x, we print $N - x$ spaces and $2x - 1$ asterisks.

We can then print the top half of the diamond using the following code.

```
for x in range(1,N+1):
      line = ""
      numAsterisks = x*2-1
      numSpaces = N - x
      for i in range(numSpaces):
            line = line + " "
      for i in range(numAsterisks):
            line = line + "*"
      print(line)
```

How do we print the bottom half of the diamond? We can **go through the line numbers in reverse** and use **the same code we used for the top half**.

```
1      *
2     ***
3    *****
4   *******
5 *********
4   *******
3    *****
2     ***
1      *
```

```
# Part 2: the bottom half of the diamond.
for x in range(N-1,0,-1):
      line = ""
      numAsterisks = x*2-1
      numSpaces = N - x
      for i in range(numSpaces):
            line = line + " "
      for i in range(numAsterisks):
            line = line + "*"
      print(line)
```

Now we just need to repeat this code for all the values of N in the input:

```
# Get the input
T = int(input())
for t in range(T):
      N = int(input())

      #Part 1: the top half
      for x in range(1,N+1):
            line = ""
            numAsterisks = x*2-1
            numSpaces = N - x
            for i in range(numSpaces):
                  line = line + " "
            for i in range(numAsterisks):
                  line = line + "*"
            print(line)

      #Part 2: the bottom half
      for x in range(N-1,0,-1):
            line = ""
            numAsterisks = x*2-1
            numSpaces = N - x
            for i in range(numSpaces):
                  line = line + " "
            for i in range(numAsterisks):
                  line = line + "*"
            print(line)
```

Solution for Problem K: Information Tracker

We have to find a way to keep track of who knows about Aldrich's plans, and who does not.

One way to do this is through a **Boolean array**. (Since array indices start at 0, we can make an array of size $N + 1$, so that we have entries for friends 1 to N).

We set everyone's entries to `false` except for friend 1 (since friend 1 is the only one who knows Aldrich's plans at the start.)

```
inputNums = [int(i) for i in input().split()]
N = inputNums[0]
M = inputNums[1]

knowledge = []
for i in range(N+1):
      knowledge.append(False)
knowledge[1] = True
```

Now we go through each of the meetings.

Note that if two friends u and v meet, they will both know Aldrich's plans only if one of u or v knows about Aldrich's plans.

```
for i in range(M):
      meeting = [int(i) for i in input().split()]
      u = meeting[0]
      v = meeting[1]
      knowledge[u] = knowledge[u] or knowledge[v]
      knowledge[v] = knowledge[u] or knowledge[v]
```

Finally, we count how many people know about Aldrich's plans.

```
count = 0
for i in range(N+1):
      if knowledge[i]:
            count += 1
print(count)
```

Below is the full solution for Problem K.

Solution:

```
inputNums = [int(i) for i in input().split()]
N = inputNums[0]
M = inputNums[1]
knowledge = []
for i in range(N+1):
      knowledge.append(False)
knowledge[1] = True

for i in range(M):
      meeting = [int(i) for i in input().split()]
      u = meeting[0]
      v = meeting[1]
      knowledge[u] = knowledge[u] or knowledge[v]
      knowledge[v] = knowledge[u] or knowledge[v]

count = 0
for i in range(N+1):
      if knowledge[i]:
            count += 1
print(count)
```

Solution for Problem M: Conway's Game of Life

This problem requires careful use of grids, with many conditional statements. When checking the neighbors of a cell, be careful to avoid going out of bounds.

```
inputNums = [int(i) for i in input().split()]
R = inputNums[0]
C = inputNums[1]

# get the input
grid = []
for row in range(R):
      inputRow = [int(i) for i in input().split()]
      grid.append(inputRow)

# counter for the answer
liveCells = 0

#for every cell in the grid:
for i in range(R):
      for j in range(C):
            # count all the neighbors
            numNeighbors = 0
            if i > 0:
                  numNeighbors += grid[i-1][j]
            if j > 0:
                  numNeighbors += grid[i][j-1]
            if i > 0 and j > 0:
                  numNeighbors += grid[i-1][j-1]
            if i+1 < R:
                  numNeighbors += grid[i+1][j]
            if j+1 < C:
                  numNeighbors += grid[i][j+1]
            if i+1 < R and j+1 < C:
                  numNeighbors += grid[i+1][j+1]
            if i > 0 and j+1<C:
                  numNeighbors += grid[i-1][j+1]
            if i+1 < R and j > 0:
                  numNeighbors += grid[i+1][j-1]

            # now we check if this cell will be alive
            if grid[i][j] == 1: #if the cell is currently alive
                  if numNeighbors==2 or numNeighbors==3:
                        liveCells += 1
            else:
                  if numNeighbors==3:
                        liveCells += 1
print("Alive cells: " + str(liveCells))
```