# Introduction

## MATA Hack 4 Good: High School Programming Competition

Saturday November 4th 2017 9:00am-1:00pm CDT
Organizer: Mid-America Technology Alliance (MATA)
Venue: The eFactory, 405 N Jefferson Ave, Springfield MO 65806

Welcome to the **MATA "Hack 4 Good" High School Programming Competition!** This event provides students with an opportunity to demonstrate and sharpen their programming skills. Continue to improve your programming skills by participating in other programming competitions.

Our High School competition **does** provide Internet access, Sample Solutions, and Hints. Please be aware that other competitions such as the ACM ICPC **do not** allow Internet access or help of any kind.

### High School Programming Competitions

- SBU HSPC (facebook.com/sbucis/) - 11/17/2017 at SBU in **Bolivar MO (30 miles)***
- UoA HSPC (hspc.csce.uark.edu) - March 2018 at UoA in **Fayetteville AR (120 miles)***
- Google Code-In (codein.withgoogle.com) - 11/28/2017
- Google Summer of Code (developers.google.com/open-source/gsoc/) - Summer 2018
- USACO (usaco.org) - Online Qualifying Contests, International Contest
- ACSL (acsl.org) - Local Qualifying Contests, National Contest

### College Programming Competitions

- CCSC-CPRPC (ccsc.org/centralplains/) 04/06/2018 at NMSU in Maryville MO
- ACM-ICPC (icpc.baylor.edu)
- HP CodeWars (hpcodewars.org)
- IEEE Xtreme (ieeextreme.org) - Annual Online Challenge
- Google Code Jam (code.google.com/codejam/) - Online Contests, International Contest

* Information about this event is attached to your packet.

Special thank you to Sherry Coker, Greg Johnson, Robin Robertson, Jason Klein, Thomas Rankin, Maranda Provance, Myke Bates, and Fredrick Lawler for their help organizing and hosting this event!

## Event Schedule

| | | |
|---|---|---|
| 9:00am | 9:30am | Team Registration and Check-In |
| 9:30am | 10:00am | Team Orientation (Welcome, Review Rules, Submit Practice Problems) |
| 10:00am | 12:00pm | Programming Competition |
| 12:00pm | 12:30pm | Lunch |
| 12:30pm | 1:00pm | Awards |

# Event Rules

The following **MATA "Hack 4 Good" High School Programming Competition** event rules have been adapted from the ACM ICPC rules (http://acmacpc.org/rules/).

## General

1. Contestants (and reserve contestant) must wear the official contest T-shirt during the contest.
2. Contestants may only use one computer per team during the contest. Teams may bring spare computers for use if primary computer fails. Computer should include a web browser and development tools they are familiar with for Java, Python 2, Python 3, C (C11), or C++ (C++14).
3. Contestants may bring printed material and reference books to the contest hall.
4. Electronic devices (mobile phones, tablets, calculators) are strictly prohibited during the contest.
5. Bags brought into the contest area may be checked.
6. Contestants are not allowed to talk to their coaches during the contest. In general, contestants are not to converse with anyone except members of their team and personnel designated by the contest director (e.g. for help submitting problems to the scoring system).
7. A team may be disqualified by the contest director for any activity that jeopardizes the contest such as dislodging extension cords, unauthorized modification of contest materials, or distracting behavior.
8. Contestants are not allowed to open problem statements until directed by the contest director.

## Contest Format

1. The contest will last exactly two hours (unless there is an unforeseen difficulty that requires extending the time.)
2. The contest problem-set is made of 12 problems. Each team attempts to solve as many problems as possible using a single computer.
3. The allowed programming languages are: Java, Python 2, Python 3, C (C11), and C++ (C++14). Contestants may use any of these allowed languages to solve each problem.
4. A contestant may submit a claim of ambiguity or error in a problem statement by submitting a clarification request. If the judges agree that an ambiguity or error exists, a clarification will be issued to all contestants. Notification of accepted runs may be suspended to keep the final results secret. Notification of rejected runs will continue until the end of the contest.
5. Solutions are judged by running them against secret test cases. The contest judges are solely responsible for determining the correctness of the submitted solutions and their decision is final.
6. Rejected runs are classified by the scoring system as follows:
    - **Compile Error**: The compiler returned an error when compiling the program.
    - **Runtime Error**: The program crashed due to segmentation fault, division by zero, etc.
    - **Time Limit Exceeded**: The program failed to complete and terminate within 5 seconds.
    - **Wrong Answer**: The program terminated successfully, but the output was incorrect.
    - **No Output**: The program terminated successfully, but there was no output.
7. If a run is rejected, the team is free to try again and send as many runs as they wish until the problem is solved (but make sure you understand the scoring rules).

## Problem Format

1. Every effort is made to guarantee that problems avoid dependence on detailed knowledge of a particular application area or particular programming language.
2. Each problem will require reading input from Standard Input (STDIN), and printing its output to the Standard Output (STDOUT). (Coaches: Make sure at least one team member can read data from STDIN and write data to STDOUT. Refer to Sample Solutions for each Practice Problem.)
3. Since judging is an automated process, it is mandatory for the program output to match the output format specified in the problem description.
4. The judges' secret input files will test the program on multiple cases (or datasets). The format of the input file will be designed so that multiple datasets can be included in a single text file.
5. Each problem in the contest is specified in the following sections:
    - **Problem Header:** Estimated Time, Number of Points, and Source Code Filename
    - **Description**: This section specifies the problem contestants are supposed to solve and also provides description on how the input file will be formatted as well as the format of the output of your program. Your program's output must conform to the format specified.
    - **Sample Input/Output**: Here you'll find a sample Input/Output that has successfully passed the judges' program. Your program will be tested on a different (and more complex) dataset. Just because your program passed the sample Input/Output, doesn't mean it is correct.
    - **Learn More**: Additional information about this problem, or about the sample inputs.
6. In addition, the problem statement will specify the name you should use for your program.
7. Since compilation and testing is an automated process, it is important that you follow the naming convention exactly. For example, if the problem statement states that the filename should be "prob01" then the program file must be named "prob01.java" for a **Java** program, "prob01.py2" for **Python 2**, "prob01.py3" for **Python 3**, "prob01.c" for **C**, or "prob01.cpp" for **C++**.

## Programming Your Solutions

1. Each program must be in a single source file with the name specified in the problem description; failure to meet this requirement will result in a "Syntax or Compilation Error".
   ***Note for Java programmers***: You can have more than one top-level class in a single source file. Only the main class should be declared public. All other classes should be unqualified (i.e. just "class", without "public").
2. Output will be judged using a file comparison utility. Output must be exactly as specified in the output format section: Spelling, punctuation, spacing, and case (uppercase/lowercase) are all significant (unless specified otherwise in the problem statement.)
3. Your program cannot require any intervention from the user. All input must be read from Standard Input (STDIN). If you submit a program that requires user intervention, you will likely receive a "Time Limit Exceeded" error. Contestants must NOT place a "Press any key to continue" statement at the end of their program.
4. All test cases used in judging will conform to the input specifications. **Your program does NOT need to detect invalid input** such as incorrect data types or missing values.
5. Make sure your program will compile and run locally before you upload your solution.

## Contest Scoring

1. Teams are ranked in a descending order according to the number of points they accumulate.
2. Teams who accumulate same number of points are ranked ascendingly by total time (the score).
3. The team who is first to solve a problem will receive special recognition on the scoreboard.
4. If the solution a team provides is incorrect, the team will be penalized 5 minutes for their incorrect attempt to solve the problem.
5. The total time (aka score) is the sum of the time consumed for each problem solved.
6. The time consumed for a solved problem is the time elapsed from the beginning of contest to the submittal of the accepted run; plus 5 penalty minutes for every rejected run of that problem.
7. No time is consumed for a problem that is not solved (even if there are rejected runs for it.)

## Scoring Example

Consider the following team which submitted 6 runs: three for problem A, two for problem B, and one for C. For each submitted run, the table shows the time of the submission, for which problem, and the judge's' response for that particular run. For solving problem A, the time consumed is 17 plus 2*5 for the two unsuccessful runs. For solving problem B, the time consumed is 0 since both runs were unsuccessful. For solving problem C, the time consumed is 13 minutes. So the total time for this team 17+2*5+0+13 = 40. (Notice that no penalties were added for problem B since it wasn't solved correctly.)

## Resolving Ties

Teams solving the same number of problems with the same total time are ranked by the geometric mean of the individual times for each solved problem (smaller being better) without the penalties. Any remaining ties are left unbroken.

## Judges' Decisions

Judges are solely responsible for accepting or rejecting submitted runs. In consultation with the judges, the contest director determines the winners of the contest. The contest director and judges are empowered to adjust for or adjudicate unforeseen events and conditions. Their decisions are final.

## Eligibility

1. Each student must be currently enrolled in a high school (or equivalent program) in the Southwest Missouri region or the surrounding regions.
2. Contestants compete in teams of 1-3 students.

## Awards and Recognition

- One trophy per team for the top 1st/2nd/3rd place teams.
- One balloon per team for each problem solved. Search online for "ICPC balloons" to learn more.
- Scoreboard will recognize which team is First To Solve each problem.
- Competition Results will be published online.

# Important Instructions

Read the following instructions carefully. They contain important information about how to submit your solutions for automated scoring. If you have any questions about these instructions, ask a volunteer before the contest begins.

## Program Input/Output

All program input must be read from Standard Input (STDIN) and all program output must be written to the screen via Standard Output (STDOUT).

Test your programs by saving the program input in a text file, then redirect the contents of your text file to your program at runtime. For example, a file named “problem.txt” can be redirected to STDIN of your program using syntax similar to this:

```
$ java problem < problem.txt
$ python problem.py < problem.txt
$ problem.exe < problem.txt
```

## Submitting Programs

You must submit a **single source code file** for your program to the scoring website. The scoring website will NOT accept multiple source code files or compiled/executable files.

Carefully test your program to make sure it is using the inputs to generate the desired output. The scoring website will automatically compile and test your solution using different inputs.

## Scoring Tips

Each problem is assigned a specific number of points and estimated time to complete. More difficult problems are assigned more points than less difficult problems. See Problem Difficulty chart below.

Your team will gain special recognition on the scoreboard if you are first to submit a correct solution for a problem. Solutions are scored in the order they are received.

Your team will be penalized 5 minutes if you submit an incorrect solution. Carefully test your solution before uploading to the scoring system!

## Problem Difficulty

Practice Problems 1-2: Complete as Group (10 points each)
Problems 1-4: Beginner (10 minutes each, 100 points each)
Problems 5-8: Intermediate (15 minutes each, 150 points each)
Problems 9-12: Advanced (20 minutes each, 200 points each)

# Practice Problems

The following problems are designed to ensure that each team is able to submit a test program to the scoring website, and that your program is able to read input (STDIN) from the scoring website, and that the scoring website is able to read output (STDOUT) from your program.

## Practice Problem 1: Testing Output

5 minutes, 10 points
Filename: prac01 *(e.g. prac01.c, prac01.cpp, prac01.java, prac01.py2, prac01.py3)*

**Description**
This problem will ensure that the scoring website is able to read output (STDOUT) from your program. Your program does not need to read input for this problem.

Output a single line containing the message “Hello, World!”. Be sure your output includes the **exact** same **case**, **punctuation**, and **spacing** shown in the sample output. Only use a single space unless instructions specifically tell you otherwise.

**Sample Input**

```
None
```

**Sample Output**

```
Hello, World!
```

**Hints**
Need help writing to Standard Output? See the Sample Solutions for this problem on the next page.

**Learn More**
While small test programs have existed since the development of programmable computers, the tradition of using the phrase "Hello, world!" as a test message was influenced by an example program in the seminal book The C Programming Language.

The example program from that book prints "hello, world" (without capital letters or exclamation mark), and was inherited from a 1974 Bell Laboratories internal memorandum by Brian Kernighan, Programming in C.A Tutorial:

```
#include <stdio.h>

main( )
{
        printf("hello, world\n");
}
```

https://en.wikipedia.org/wiki/”Hello,_World!”_program
Prepared by Jason Klein. Adapted from CodeWars.

# Practice Problem 1: Sample Solutions

See below for **Practice Problem 1** solutions written in **Java, Python 2, Python 3, C, and C++**.

Each solution demonstrates how to output strings to Standard Output (STDOUT). **Each solution has been tested with our scoring system.**

Each solution also demonstrates how to compile the program and how to read input from a text file during runtime. Each program will generate the **Sample Output** shown below.

**Sample Output**

```
Hello, World!
```

**Sample Java Code (prac01.java)**

```
import java.io.*;

public class prac01 {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Compile, Run

```
javac prac01.java
java prac01
```

**Sample Python2 Code (prac01.py2)**

```
print "Hello, World!"
```

Run

```
python2 prac01.py2
```

**Sample Python3 Code (prac01.py3)**

```
print("Hello, World!")
```

Run

```
python3 prac01.py3
```

**Check Python Version**

```
python -V
```

This will output the exact version, such as “Python 2.7.13” for **Python 2** or “Python 3.5.3” for **Python 3**.

**Sample C Code (prac01.c)**

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

We recommend using the C11 standard. Be sure to return 0 in your code. A missing or different return statement might result in invalid exit codes that the scoring system interprets as a Runtime Error.

Compile, Run

```
gcc prac01.c -o prac01
./prac01
```

**Sample C++ Code (prac01.cpp)**

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

We recommend using the C++14 standard. Be sure to return 0 in your code. A missing or different return statement might result in invalid exit codes that the scoring system interprets as a Runtime Error.

Compile, Run

```
g++ prac01.cpp -o prac01
./prac01
```

# Practice Problem 2: Testing Input and Output

5 minutes, 10 points
Filename: prac02 *(e.g. prac02.c, prac02.cpp, prac02.java, prac02.py2, prac02.py3)*

**Description**
This problem will ensure that your program is able to read input (STDIN) from the scoring website, and that the scoring website is able to read output (STDOUT) from your program.

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain two words separated by a space.

For each line of input containing NOUN and VERB, output a sentence constructed as “NOUN is VERB today!” Be sure that your output includes the **punctuation** shown in the sample output.

**Sample Input**

```
3
Mitchell coding
Danese testing
Larry debugging
```

**Sample Output**

```
Mitchell is coding today!
Danese is testing today!
Larry is debugging today!
```

**Hints**
Need help writing to Standard Output? See the Sample Solutions for this problem on the next page.

**Learn More**
Mitchell Baker was instrumental in the creation of the Mozilla Foundation, the organization that oversees development of the Firefox browser and other products. Today, she is chairperson of both the Mozilla Foundation and Mozilla Corporation.

Danese Cooper currently serves as chairperson of the Node.js Foundation. She also works in an open source role at PayPal. Previously, she served as CTO of the Wikimedia Foundation.

Larry Augustin is CEO of SugarCRM. He helped coin the term “Open Source”. In 1993 he founded VA Linux (now SourceForge, NASDAQ:LNUX), where he served as CEO until 2002.

# Practice Problem 2: Sample Solutions

See below for **Practice Problem 2** solutions written in **Java, Python 2, Python 3, C, and C++**.

Each solution demonstrates how to read integers and strings from Standard Input (STDIN) and output strings to Standard Output (STDOUT). **Each solution has been tested with our scoring system.**

Each solution also demonstrates how to compile the program and how to read input from a text file during runtime. When using the following **Input**, each program will generate the following **Output**.

**Sample Input (prac02.in)**

```
3
Mitchell coding
Danese testing
Larry debugging
```

**Sample Output**

```
Mitchell is coding today!
Danese is testing today!
Larry is debugging today!
```

**Sample Java Code (prac02.java)**

```
import java.io.*;

public class prac02 {
    public static void main(String[] args) throws IOException {
        BufferedReader reader
            = new BufferedReader(new InputStreamReader(System.in));

        int numCases = Integer.parseInt(reader.readLine());

        while (numCases > 0) {
            --numCases;
            String line[] = reader.readLine().trim().split("\\s+");
            String noun = line[0];
            String verb = line[1];
            System.out.println(noun + " is " + verb + " today!");
        }
    }
}
```

**Compile, Run with Input File**

```
javac prac02.java
java prac02 < prac02.in
```

**Sample Python2 Code (prac02.py2)**

```
import sys

lines = sys.stdin.readlines()
num_cases = int(lines[0])

case = 1
while case < num_cases + 1:
    noun, verb = lines[case].split()
    print "%s is %s today!" % (noun, verb)
    case = case + 1
```

**Run with Input File**

```
python2 prac02.py2 < prac02.in
```

**Sample Python3 Code (prac02.py3)**

```
import sys

lines = sys.stdin.readlines()
num_cases = int(lines[0])

case = 1
while case < num_cases + 1:
    noun, verb = lines[case].split()
    print("%s is %s today!" % (noun, verb))
    case = case + 1
```

**Run with Input File**

```
python3 prac02.py3 < prac02.in
```

**Sample C Code (prac02.c)**

```
#include <stdio.h>

int main(int argc, char **argv) {
    int numCases;
    char noun[99 + 1]; /* + '\0' */
    char verb[99 + 1]; /* + '\0' */

    scanf("%d", &numCases);
    while (numCases) {
        --numCases;
        scanf("%s %s", noun, verb);
        printf("%s is %s today!\n", noun, verb);
    }
    return 0;
}
```

We recommend using the C11 standard. Be sure to return 0 in your code. A missing or different return statement might result in invalid exit codes that the scoring system interprets as a Runtime Error.

**Compile, Run with Input File**

```
gcc prac02.c -o prac02
./prac02 < prac02.in
```

**Sample C++ Code (prac02.cpp)**

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char **argv) {
    int numCases;
    string name, verb;

    cin >> numCases;
    while (numCases) {
        --numCases;
        cin >> name >> verb;
        cout << name << " is " << verb << " today!" << endl;
    }
    return 0;
}
```

We recommend using the C++14 standard. Be sure to return 0 in your code. A missing or different return statement might result in invalid exit codes that the scoring system interprets as a Runtime Error.

**Compile, Run with Input File**

```
g++ prac02.cpp -o prac02
./prac02 < prac02.in
```

# Beginner Problems

## Problem 1: Unit Conversion

10 minutes, 100 points
Filename: prob01 *(e.g. prob01.c, prob01.cpp, prob01.java, prob01.py2, prob01.py3)*

**Description**
We need to calculate the weight of items to be shipped in kilograms (KG). One kilogram (KG) equals 2.2 pounds (LB).

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain number of gallons (GAL) and weight of liquid in pounds per gallon (LB/GAL).

Multiply the number of gallons (GAL) times the weight of the liquid (LB/GAL) to determine total weight in pounds (LB). Then convert the total weight from pounds (LB) to kilograms (KG) by dividing pounds (LB) by 2.2.

GAL x LB/GAL = LB, then LB / 2.2 = KG

Example:

144 gal x 8.34 LB/gal = 1200.96 gal, then 1200.96 LB / 2.2 KG/LB = 545.89 KG

Output the weight in Kilograms (KG) for each shipment. Round your result to the nearest hundredth of a kilogram (two decimal places). Always display two decimal places (e.g. 432.1 = 432.10).

**Sample Input**

```
3
144 8.34
288 7.6
250 7.344
```

**Sample Output**

```
545.89
994.91
834.55
```

**Learn More**
Liquid density varies by type of liquid. Water weighs 8.34 pounds per gallon, while olive oil only weighs 7.6 pounds per gallon, and petroleum oil weighs even less at 7.344 pounds per gallon.

# Problem 2: Profit Margin

10 minutes, 100 points
Filename: prob02 *(e.g. prob02.c, prob02.cpp, prob02.java, prob02.py2, prob02.py3)*

**Description**
We are helping someone sell their shirts at the local market. Given daily costs, shirt costs, shirt sales price, and number of shirts sold, they need our help calculating their profit margin for each day.

Begin by calculating per shirt profit:
    $4.75 price - $1.86 cost = $2.89 profit
Then calculate shirt profit for the day:
    13 shirts at $2.89 profit per shirt = $37.57 shirt profit
Finally, subtract the daily booth cost from the shirt profit to determine profit margin for the day:
    $37.57 shirt profit - $20.00 daily cost = $17.57 profit margin for the day

The first line of input will contain a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain the daily booth cost, the per shirt cost, the per shirt sale price, and the quantity of shirts sold at the market that day.

Output the profit margin for the day. Round to two decimal places. Always display two decimal places (e.g. 25.1 = 25.10).

**Sample Input**

```
3
20 1.86 4.75 13
20 1.86 4.75 42
20 1.86 4.75 52
```

**Sample Output**

```
17.57
101.38
130.28
```

**Hints**
You can safely assume the shirts will always be sold for more than they cost.

Make sure your program can return a negative profit margin if the daily shirt profits are less than the daily booth costs (e.g. negative profit of 1.23 would appear as -1.23)

# Problem 3: Equation

10 minutes, 100 points
Filename: prob03 *(e.g. prob03.c, prob03.cpp, prob03.java, prob03.py2, prob03.py3)*

**Description**
We need to convert a weather forecast from Celsius (C) and KM/H to Fahrenheit (F) and MPH.

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain the daily high temperature in Celsius (C), daily low temperature in Celsius (C), and wind speed in Kilometers/Hour (KM/H).

Use these formulas to convert temperatures and wind speeds:

Celsius to Fahrenheit: C x 9/5 + 32 = F (e.g. 14 x 9/5 + 32 = 57.2, rounded to 57)
KM to MI: KM / 1.609344 = MI (e.g. 19 / 1.6093 = 11.8064, rounded to 12)

Each line of output should contain the high and low temperatures in Fahrenheit, followed by the wind speed in Miles/Hour. Round your results to nearest integer. Be sure to append the letter “F” after each temperature and the string “MPH” after each wind speed, as shown in the sample output.

**Sample Input**

```
3
14 7 19
19 9 13
26 12 11
```

**Sample Output**

```
57F 45F 12MPH
66F 48F 8MPH
79F 54F 7MPH
```

**Learn More**
There are two main temperature scales: °F, the Fahrenheit Scale (used in the US), and °C, the Celsius Scale (part of the Metric System, used in most other countries)

They both measure temperature, but use different numbers:

Boiling water (at normal pressure) measures 100° in Celsius, but 212° in Fahrenheit
As water freezes it measures 0° in Celsius, but 32° in Fahrenheit

The scales rise at a different rate (100 vs 180), so we will need to multiply or divide by 100/180.
The scales start at a different number (0 vs 32), so we will need to add or subtract 32.

To convert from Celsius to Fahrenheit: first multiply by 180/100 (or 9/5), then add 32
To convert from Fahrenheit to Celsius: first subtract 32, then multiply by 100/180 (or 5/9)

https://www.mathsisfun.com/temperature-conversion.html#explanation
Prepared by Jason Klein

# Problem 4: Divisibility Rules

10 minutes, 100 points
Filename: prob04 *(e.g. prob04.c, prob04.cpp, prob04.java, prob04.py2, prob04.py3)*

**Description**
Determine if the given positive integer is divisible by 3 using the following divisibility rule.

Sum each digit of the integer. Determine if the sum of digits is a multiple of 3. If so, the original positive integer is divisible by 3. (e.g. Given 12345, sum of digits would be 1+2+3+4+5=15. Since 15 is a multiple of 3, we know 12345 is divisible by 3).

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain a single positive integer.

Output the original integer, the sum of the digits, and whether or not the integer is divisible by 3. Be sure to return “YES” or “NO” in uppercase.

**Sample Input**

```
4
123
12345
1234567
12345678
```

**Sample Output**

```
123 6 YES
12345 15 YES
1234567 28 NO
12345678 36 YES
```

**Learn More**
A divisibility rule is a heuristic for determining whether a positive integer can be evenly divided by another (i.e. there is no remainder left over). For example, determining if a number is even is as simple as checking to see if its last digit is 2, 4, 6, 8 or 0. Multiple divisibility rules applied to the same number in this way can help quickly determine its prime factorization without having to guess at its prime factors.

https://brilliant.org/wiki/divisibility-rules/
Prepared by Jason Klein

# Intermediate Problems

## Problem 5: Search and Replace

15 minutes, 150 points
Filename: prob05 *(e.g. prob05.c, prob05.cpp, prob05.java, prob05.py2, prob05.py3)*

**Description**
You will be given a word, a search string, and a replace string. Perform the search/replace on the original word and output the new word.

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain a word, a search string, and a replace string. The original word and search/replace strings will only contain uppercase letters. The word and strings will NOT contain any numbers, spaces, or other special characters.

Each line of output should contain the new word. If the search string is not found in the original word, output the original word without making any changes.

**Sample Input**

```
3
OFTEN FT P
SOURCED ED E
HARDWARE HARD SOFT
```

**Sample Output**

```
OPEN
SOURCE
SOFTWARE
```

# Problem 6: Calculate Check Digit

15 minutes, 150 points
Filename: prob06 *(e.g. prob06.c, prob06.cpp, prob06.java, prob06.py2, prob06.py3)*

**Description**
Every bank is assigned a 9-digit routing number (8 digits followed by a check digit). Given the first 8 digits (e.g. 01234567), calculate the check digit and output the 9-digit routing number.

Step 1: $(d_1 \times 3) + (d_2 \times 7) + (d_3 \times 1) + (d_4 \times 3) + (d_5 \times 7) + (d_6 \times 1) + (d_7 \times 3) + (d_8 \times 7) = \text{Total}$
*Example: (0 x 3) + (1 x 7) + (2 x 1) + (3 x 3) + (4 x 7) + (5 x 1) + (6 x 3) + (7 x 7) = 118*

Step 2: Using the Total (118), take the next highest number that is evenly divisible by 10 and subtract the total from this to produce the proper checksum digit (120 - 118 = 2 in this case). If the total is already an even multiple of ten, say 110, the checksum would simply be zero.

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain an 8 digit integer.

The output should contain the 9 digit integer. Be sure to pad the beginning of the number with zeroes as necessary.

**Sample Input**

```
3
01234567
11111111
50505050
```

**Sample Output**

```
012345672
111111118
505050500
```

**Learn More**
An ABA routing transit number (ABA RTN) is a 9-digit code which appears on the bottom of US cheques to identify the financial institution on which it was drawn. The ABA RTN was originally designed by the American Bankers Association in 1910 to facilitate the sorting, bundling, and shipment of paper cheques back to the check writer's account. The system has been expanded to accommodate ACH and Wire payment methods.

To verify a routing number, the following condition must hold (where $d_n$ represents a digit):

$$3(d_1 + d_4 + d_7) + 7(d_2 + d_5 + d_8) + 1(d_3 + d_6 + d_9) \bmod 0 = 0$$

The ABA RTN is necessary for the Federal Reserve Banks to process Fedwire funds transfers, and by Automated Clearing House to process direct deposits, bill payments, and other transfers.

https://en.wikipedia.org/wiki/Routing_transit_number
http://www.brainjar.com/js/validation/
Prepared by Jason Klein. Adapted from CodeWars and St. Bonaventure University

# Problem 7: Identify Closest Guess

15 minutes, 150 points
Filename: prob07 *(e.g. prob07.c, prob07.cpp, prob07.java, prob07.py2, prob07.py3)*

**Description**
You are hosting a contest to guess the correct number of pennies in a large jar. Given the actual number of pennies and each guess with the first name of the person who submitted the guess, list the first name(s) that guessed closest to the actual amount.

The first line of input will include a single integer that represents the correct number of items. The second line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain a guess and a name.

The output should contain the name(s) of the closest guesses on a single line. If there are multiple matches closest matches (e.g. a tie), return each name on the same line separated by spaces. Names should be ordered same as they appear in the original list of guesses.

**Sample Input**

```
480
4
90 Karissa
400 Monica
560 Ben
173 Jade
```

**Sample Output**

```
Monica Ben
```

**Sample Input**

```
362
5
123 Andrew
456 Dylan
321 Adrienne
400 Caleb
314 Miles
```

**Sample Output**

```
Caleb
```

# Problem 8: Pangram

15 minutes, 150 points
Filename: prob08 *(e.g. prob08.c, prob08.cpp, prob08.java, prob08.py2, prob08.py3)*

**Description**
A sentence that uses every letter of the alphabet is called a pangram. A “perfect” pangram only uses each letter once. Given a sentence, determine if the sentence is a perfect pangram (each letter of alphabet once), a pangram, or neither.

Each line contains a sentence containing no more than 60 characters. Every sentence will be formatted in uppercase.

Return “PERFECT”, “PANGRAM”, or “NEITHER” followed by a colon, then a space, then the original sentence. Stop processing when you receive a line that contains only a period.

**Sample Input**

```
YOU CAN'T TEACH AN OLD DOG NEW TRICKS
SPHINX OF BLACK QUARTZ, JUDGE MY VOW
MR JOCK, TV QUIZ PHD, BAGS FEW LYNX
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
IF YOU CAN'T STAND THE HEAT, GET OUT OF THE KITCHEN
THE FIVE BOXING WIZARDS JUMP QUICKLY
.
```

**Sample Output**

```
NEITHER: YOU CAN'T TEACH AN OLD DOG NEW TRICKS
PANGRAM: SPHINX OF BLACK QUARTZ, JUDGE MY VOW
PERFECT: MR JOCK, TV QUIZ PHD, BAGS FEW LYNX
PANGRAM: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
NEITHER: IF YOU CAN'T STAND THE HEAT, GET OUT OF THE KITCHEN
PANGRAM: THE FIVE BOXING WIZARDS JUMP QUICKLY
```

**Learn More**
The best-known English pangram is "The quick brown fox jumps over the lazy dog." It has been used since at least the late 19th century, was utilized by Western Union to test communication equipment for accuracy and reliability, and is now used by a number of computer programs (most notably the font viewer built into Microsoft Windows) to display computer fonts.

https://en.wikipedia.org/wiki/Pangram
Prepared by Jason Klein. Adapted from CodeWars

# Advanced Problems

## Problem 9: Scientific Notation

20 minutes, 200 points
Filename: prob09 *(e.g. prob09.c, prob09.cpp, prob09.java, prob09.py2, prob09.py3)*

**Description**
Scientific notation is a way of expressing numbers that are too big or too small to be conveniently written in decimal form. It is commonly used by scientists, mathematicians and engineers, in part because it can simplify certain arithmetic operations.

Here is an example of a number written in scientific notation and it's decimal equivalent.

```
6.9105 x 10^2 = 691.05
```

Write a program that will read in a scientific number as a coefficient m and an exponent, then calculate and output the equivalent decimal value. Your program must convert the coefficient (M) and the exponent (E) to decimal w/ two places.

Each line of input will contain a decimal coefficient and a positive or negative integer exponent. Stop processing the input when you receive zero values for both the coefficient and the exponent.

Each line of output should contain the decimal equivalent value. Return two decimal places. Round the result as necessary (e.g. 8675.309 = 8675.31). Pad zeroes as necessary (e.g. 470 = 470.00).

**Sample Input**

```
6.9105 2
113.8 -2
4.7 2
867.5309 1
0 0
```

**Sample Output**

```
691.05
1.14
470.00
8675.31
```

**Learn More**
In scientific notation all numbers are written in the following form, where the coefficient m is any real number, and the exponent n is an integer.

$m \times 10^n$ (m times ten raised to the power of n)

https://en.wikipedia.org/wiki/Scientific_notation
Prepared by Jason Klein. Adapted from CodeWars

# Problem 10: Common Letters

20 minutes, 200 points
Filename: prob10 *(e.g. prob10.c, prob10.cpp, prob10.java, prob10.py2, prob10.py3)*

**Description**
Write a program that can determine the set of letters shared in common among three words.
Common letters (TEST MEANT TIME = TE)

The first line of input will include a single integer that indicates how many additional lines of input need to be read. Each additional line of input will contain three words separated by one or more spaces.

Each line of output should contain the common letter(s) for the three words. If there are multiple common letters for the three words, return each letter on the same line. The common letters should appear in the same order as they appear in the first word. Each common letter should only appear once per line of output. If the words in a line of input do not have any common letters, output a blank line.

**Sample Input**

```
3
ENGINEER   DESIGNER   RESEARCHING
OPEN       SOURCE     SOFTWARE
ENCODE     DOCKER     LOCKED
```

**Sample Output**

```
ENGIR
OE
ECOD
```

**Hints**
Words may be separated by one or more **Space** and/or **Tab** characters. When reading the three words from each line, split the string by one or more whitespaces. Python “split()” and C++ “cin” handle this automatically, but Java “split()” requires a special whitespace regular expression. See examples below.

Python

```
word1, word2, word3 = lines[case].split()
```

C++

```
cin >> word1 >> word2 >> word3
```

Java

```
String[] words = input.nextLine().split("\\s+");
```

# Problem 11: Combinations

20 minutes, 200 points
Filename: prob11 *(e.g. prob11.c, prob11.cpp, prob11.java, prob11.py2, prob11.py3)*

**Description**
Vendor provides packages of 7, 11, and 13 items. The cost to prepare each package is about the same. Determine the fewest packages required to fulfil an order for a specific number of items.

For example, 42 items can be packed into 1 package of 7, 2 packages of 11, and 1 package of 13, for a total of 4 packages (42 1 2 1 4).

Each line of input contains an integer representing the number of items ordered. Each input value will be a positive integer less than 1000. The last line of input will contain a zero. No other lines will contain a zero. When you receive a zero value, you can stop processing the input.

Each line of output should contain the original number of items, followed by the number of packages of 7, the number of packages of 11, the number of packages of 13, and the total number of packages. If the order cannot be fulfilled from any combination of packages, return the original number of items ordered with no other data (e.g. 27).

**Sample Input**

```
42
55
27
88
0
```

**Sample Output**

```
42 1 2 1 4
55 1 2 2 5
27
88 2 2 4 8
```

**Hints**
A simple (but inefficient) “brute force” approach to this problem would be to loop through each possible combination of packages, trying to fill the largest packages first, and returning the first combination of packages that contain the exact number of items ordered. The scoring system will accept a program that uses this approach, as long as your program does not run for more than 5 seconds on the scoring server.

# Problem 12: Polite Numbers

20 minutes, 200 points
Filename: prob12 *(e.g. prob12.c, prob12.cpp, prob12.java, prob12.py2, prob12.py3)*

**Description**
In number theory, a polite number is a positive integer that can be written as the sum of two or more consecutive positive integers. Other positive integers that **cannot** be written as the sum of two or more consecutive positive integers are **not polite**.

**Examples**

14 = 2 + 3 + 4 + 5, so 14 is polite
15 = 7 + 8, so 15 is polite
16 is not polite (no sum of two or more consecutive numbers equal 16)
17 = 8 + 9, so 17 is polite
18 = 5 + 6 + 7, so 18 is polite
19 = 9 + 10, so 19 is polite

Each line of input will include a single integer that needs to be tested to determine whether or not it is a polite number. Each input value will be a positive integer less than or equal to 1000. The last line of input will contain a zero. No other lines will contain a zero. When you receive a zero value, you can stop processing the input.

Each line of output should contain the original integer, followed by " is polite" or " is not polite".

**Sample Input**

```
14
15
16
17
18
19
0
```

**Sample Output**

```
14 is polite
15 is polite
16 is not polite
17 is polite
18 is polite
19 is polite
```

**Hints**
A simple (but inefficient) "brute force" approach to this problem would be to loop through each possible combination of sums, checking to see if the sum matches the input. If no combinations of sums equal the input, the number is not polite. The scoring system will accept a program that uses this approach, as long as your program does not run for more than 5 seconds on the scoring server.

https://en.wikipedia.org/wiki/Polite_number
Prepared by Jason Klein. Adapted from CodeWars

# Index