CS1010E: Programming Methodology

Take Home Lab 1: Operations

TBA

Preliminary: Time Elapsed [#1]

Problem Description

The amount of time elapsed in a day can be measured in either seconds, minutes, or hours. One minute is equivalent to 60 seconds and one hour is equivalent to 60 minutes.

Final Objective

Given the number of hours, minutes, and seconds elapsed, compute the amount of time elapsed in seconds.

Example

Imagine that 2 hours, 13 minutes, and 35 seconds has elapsed. The amount of time elapsed is simply: $35 + (13 \times 60) + (2 \times 60 \times 60) = 35 + 780 + 7200 = 8015$ seconds.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ $0 \leq$ `hour` ≤ 24 (*the time elapsed in hours*)
- ▷ $0 \leq$ `min` ≤ 59 (*the time elapsed in minutes*)
- ▷ $0 \leq$ `sec` ≤ 59 (*the time elapsed in seconds*)

Restrictions

The following restriction(s) is/are imposed on the solution:

- ▷ You cannot use selection statements such as (*but not limited to*) **`if`**, **`switch`**, or `?:` operator
- ▷ You cannot use repetition statements such as (*but not limited to*) **`while`**, **`do-while`**, or **`for`**
- ▷ You cannot use `<math.h>` library

Tasks

The problem is split into 1 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/1

Write a program that reads **three (3)** `integer` numbers corresponding to the hour, minutes, and seconds elapsed and print the amount of time elapsed in seconds as `integer` with additional [`newline`] at the end.

Sample Run:

Inputs:	Outputs:
`2 13 35`	`8015↩`

Sample Run:

Inputs:	Outputs:
`1 0 0`	`3600↩`

Sample Run:

Inputs:	Outputs:
`2 0 0`	`7200↩`

Save your program in the file named `time_elapsed1.c`. Submit your program to CodeCrunch.

Easy: Temperature [#2]

Problem Description

"A temperature is an objective comparative measurement of hot or cold. It is measured by a thermometer. Several scales and units exist for measuring temperature, the most common being Celsius (denoted C; formerly called centigrade), Fahrenheit (denoted F), and, especially in science, Kelvin (denoted K)." – Wikipedia

Formula 1 shows a formula for converting Kelvin to Celsius and Formula 2 shows a formula for converting Kelvin to Fahrenheit. In the formula, K is the temperature in Kelvin, C is the temperature in Celsius, and F is the temperature in Fahrenheit.

$$C = K - 273.15 \tag{1}$$

$$F = (K - 273.15) \times 1.8 + 32 \tag{2}$$

Final Objective

Given a temperature in Kelvin, compute the equivalent temperature in Celsius and Fahrenheit.

Example

Table 1 shows several temperatures in the three measurements.

Kelvin	Celsius	Fahrenheit
59	-214.15	-353.47
233.15	-40.00	-40.00
270	-3.15	26.33
1000	726.85	1340.33

Table 1: Some temperature values in Kelvin, Celsius, and Fahrenheit.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ -270 $\leq$ K $\leq$ 10000 (*the temperature in Kelvin*)

Restrictions

The following restriction(s) is/are imposed on the solution:

- ▷ You cannot use selection statements such as (*but not limited to*) **if**, **switch**, or ?: operator
- ▷ You cannot use repetition statements such as (*but not limited to*) **while**, **do-while**, or **for**
- ▷ You cannot use <**math**.**h**> library

Tasks

The problem is split into 1 task(s). In the sample run, please note the following:

- ↩ is the *invisible* [**newline**] character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/1

Write a program to read a single **real** number corresponding to the temperature in Kelvin and prints **two (2)** **real** numbers up to **two (2)** decimal places corresponding to the temperature in Celsius and Fahrenheit respectively followed by a [**newline**] at the end.

Sample Run:

Inputs:	Outputs:
59	-214.15 -353.47↩

Sample Run:

Inputs:	Outputs:
233.15	-40.00 -40.00↩

Sample Run:

Inputs:	Outputs:
270	-3.15 26.33↩

Sample Run:

Inputs:	Outputs:
1000	726.85 1340.33↩

Save your program in the file named `temperature1.c`. Submit your program to CodeCrunch.

Easy: Check Digit [#3]

Problem Description

"A check digit is a form of redundancy check used for error detection on identification numbers, such as bank account numbers, which are used in an application where they will at least sometimes be input manually. It is analogous to a binary parity bit used to check for errors in computer-generated data. It consists of one or more digits computed by an algorithm from the other digits (or letters) in the sequence input.

"With a check digit, one can detect simple errors in the input of a series of characters (usually digits) such as a single mistyped digit or some permutations of two successive digits." – Wikipedia

NUS student ID employs this checking. Since the new student ID is easier to check – *hence, easier for error to occur* – we will use the old NUS student ID instead. Old NUS student ID has the form of `UXX0XXXX` where `U` means that the student is an undergraduate (*there are scheme for graduate students and non-graduating students, but we will ignore those*), `X` is a single digit number, and `0` is the number zero. Note how the third-digit from the left is always zero.

We will number the index of the digit with the leftmost digit at index 1. Ignoring the leading `U`, the algorithm to generate the check digit is as follows:

1. Discard the digits at index 1 and 3.
2. Multiply the digit at index 4 by 3, the digit at index 6 by 2, and the last digit at index 7 by 7.
3. Let S be the sum of all the digits after multiplication.
4. The check-digit is S `mod` 13.

Final Objective

Given an old NUS student ID, compute the check digit.

Example

Consider a student with ID `U0707005`. Legend has it that this student ID belongs to one of the most brilliant mind in NUS. The check digit is 11. The intermediate computation of the check digit is shown in Table 2.

	`0`	`7`	`0`	`7`	`0`	`0`	`5`
Step #1		7		7	0	0	5
Step #2		7		7×3	0	0×2	5×7
Step #3		7		21	0	0	35

Table 2: Values of some conversions from metric to imperial units.

The sum of the number is $S = 7 + 21 + 0 + 0 + 35 = 63$. Thus, the check digit is S `mod`$13 = 11$.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ 0000000 $\leq$ `id` $\leq$ 9999999 (*old NUS student ID*)
- ▷ `id` may or may not have leading zeroes

Restrictions

The following restriction(s) is/are imposed on the solution:

- ▷ You cannot use selection statements such as (*but not limited to*) **`if`**, **`switch`**, or `?:` operator
- ▷ You cannot use repetition statements such as (*but not limited to*) **`while`**, **`do-while`**, or **`for`**
- ▷ You cannot use `<math.h>` library

Tasks

The problem is split into 2 task(s). In the sample run, please note the following:

- ↩ is the *invisible* [**newline**] character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/2

Write a program to read a single **integer** number corresponding to the student ID number and print all the individual digits separated by a single [**space**] except after the last number and followed by a [**newline**].

Sample Run:

Inputs:	Outputs:
`707005 | no leading zeroes`	`0 7 0 7 0 0 5↩`

Sample Run:

Inputs:	Outputs:
`0707005 | with leading zeroes`	`0 7 0 7 0 0 5↩`

Save your program in the file named `matric1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp matric1.c matric2.c
```

Task 2/2

Write a program to read a single **integer** number corresponding to the student ID number and print a single **integer** number corresponding to the check digit followed by a [**newline**] at the end.

Sample Run:

Inputs:	Outputs:
`707005 | no leading zeroes`	`11↩`

Sample Run:

Inputs:	Outputs:
`0707005 | with leading zeroes`	`11↩`

Save your program in the file named `matric2.c`. Submit your program to CodeCrunch.

Medium: Matrix Determinant [#4]

Problem Description

"In linear algebra, the determinant is a useful value that can be computed from the elements of a square matrix. The determinant of a matrix A is denoted $det(A)$, $detA$, or $\|A\|$. It can be viewed as the scaling factor of the transformation described by the matrix." – Wikipedia

The determinant of a 2×2 matrix is trivial. It can be computed using Formula 3.

$$det(\begin{vmatrix} a & b \\ c & d \end{vmatrix}) = ad - bc \tag{3}$$

The determinant of a 3×3 matrix is more complicated. To compute the determinant of a 3×3 matrix, you have to compute the determinant of a 2×2 matrix thrice as shown in Formula 4.

$$det(\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}) = a \times det(\begin{vmatrix} e & f \\ h & i \end{vmatrix}) - b \times det(\begin{vmatrix} d & f \\ g & i \end{vmatrix}) + c \times det(\begin{vmatrix} d & e \\ g & h \end{vmatrix}) \tag{4}$$

Final Objective

Given a 3×3 matrix, compute its determinant.

Example

Verify the correctness of the computation below yourself.

$$det(\begin{vmatrix} 1 & 2 & 3 \\ 1.5 & 2.5 & 3.5 \\ 3 & 2 & 4 \end{vmatrix}) = -1.5 \tag{5}$$

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ `-1000` $\leq$ `a - i` $\leq$ `1000` (*the value in the matrix*)

Restrictions

The following restriction(s) is/are imposed on the solution:

▷ You cannot use selection statements such as (*but not limited to*) **`if`**, **`switch`**, or `?:` operator

▷ You cannot use repetition statements such as (*but not limited to*) **`while`**, **`do-while`**, or **`for`**

▷ You cannot use `<math.h>` library

Tasks

The problem is split into 3 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/3

Write a program to read **nine (9)** `real` numbers corresponding to the element in the matrix and print the 3×3 matrix as a **two (2)** decimal place `real` numbers with a single `[space]` between values and a `[newline]` after each line including the last line.

Sample Run:

Inputs:	Outputs:

`1 2 3`
`1.5 2.5 3.5`
`3 2 4`
	`1.00 2.00 3.00↩`
	`1.50 2.50 3.50↩`
	`3.00 2.00 4.00↩`

Save your program in the file named `matrix1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp matrix1.c matrix2.c
```

Task 2/3

Write a program to read **nine (9)** `real` numbers corresponding to the element in the matrix and print **three (3)** `real` numbers up to **two (2)** decimal places corresponding to $det(\begin{vmatrix} e & f \\ h & i \end{vmatrix})$, $det(\begin{vmatrix} d & f \\ g & i \end{vmatrix})$, and $det(\begin{vmatrix} d & e \\ g & h \end{vmatrix})$ respectively each separated by a single `[space]` and a `[newline]` at the end of the line.

Sample Run:

Inputs:	Outputs:

`1 2 3`
`1.5 2.5 3.5`
`3 2 4`
	`3.00 -4.50 -4.50↩`

Save your program in the file named `matrix2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp matrix2.c matrix3.c
```

Task 3/3

Write a program to read **nine (9)** `real` numbers corresponding to the element in the matrix and print a single `real` number up to **two (2)** decimal places corresponding to $det(\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix})$ with a `[newline]` at the end of the line.

Sample Run:

Inputs:	Outputs:

`1 2 3`
`1.5 2.5 3.5`
`3 2 4`
	`-1.50↩`

Save your program in the file named `matrix3.c`. Submit your program to CodeCrunch.

Hard: Finding Non-Duplicates [#5]

Problem Description

Exclusive OR (or XOR) operation is a very peculiar operation denoted as $\oplus$. It has a special property that if two numbers are equal, the result is always zero. Additionally, any number x when XOR'ed with 0 will always result in that number again (*i.e.,* $x \oplus 0 = x$). In C code, it is represented by the operator ^ (*i.e., [shift + 6]*). We can abuse this for our purpose.

Final Objective

Given a 9 digit numbers such that all but one number is duplicated, find the non-duplicated number without using any selection statement such as **`if`** or **`switch`**.

Example

Consider the number `152442315`. The non-duplicated number is 3. Find other examples yourself.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ $0 \leq$ `num` ≤ 999999999 (*the number to be checked*)
- ▷ `num` only have a single non-duplicated number

Restrictions

The following restriction(s) is/are imposed on the solution:

- ▷ You cannot use selection statements such as (*but not limited to*) **`if`**, **`switch`**, or `?:` operator
- ▷ You cannot use repetition statements such as (*but not limited to*) **`while`**, **`do-while`**, or **`for`**
- ▷ You cannot use **`<math.h>`** library

Tasks

The problem is split into 3 task(s). In the sample run, please note the following:

- ↩ is the *invisible* **`[newline]`** character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/3

Write a program to read a single **`integer`** number and print exactly **nine (9)** **`integer`** numbers corresponding to each digit of the number each separated by a single **`[space]`** and a **`[newline]`** at the end of the line.

Sample Run:

Inputs:	Outputs:
152442315	1 5 2 4 4 2 3 1 5↩

Save your program in the file named `duplicate1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp duplicate1.c duplicate2.c
```

Task 2/3
Write a program to read a single **integer** number and print a single **integer** corresponding to the sum of all the digits terminated by a [**newline**].
Sample Run:

Inputs:	Outputs:
152442315	27↩

Save your program in the file named `duplicate2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:
`cp duplicate2.c duplicate3.c`

Task 3/3
Write a program to read a single **integer** number and print a single **integer** corresponding to the non-duplicated digit terminated by a [**newline**].
Sample Run:

Inputs:	Outputs:
152442315	3↩

Save your program in the file named `duplicate3.c`. Submit your program to CodeCrunch.

Hard: 10 Jars Puzzle [#6]

Problem Description

There are 10 jars, each with 10 marbles. All the marbles are 10 grams, except for one jar which has all 9 gram marbles. If we label the jar from 1 to 10, find the jar which has all 9 gram marbles.

Final Objective

Given the weight of the marble in every jar, find the unique jar without using any selection statement such as **`if`** or **`switch`**.

Example

No example, because by inspecting the input, we (*as a human*) can easily know which jar it is. Furthermore, since this is a programming problem, I have to give you the weight. Try to solve it yourself by assuming you do not know the weight and you can only weigh once (*to simulate no selection statement*).

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ $9 \leq$ `jar1` ≤ 10 (*weight of marbles in jar #1*)
- ▷ $9 \leq$ `jar2` ≤ 10 (*weight of marbles in jar #2*)
- ▷ $9 \leq$ `jar3` ≤ 10 (*weight of marbles in jar #3*)
- ▷ $9 \leq$ `jar4` ≤ 10 (*weight of marbles in jar #4*)
- ▷ $9 \leq$ `jar5` ≤ 10 (*weight of marbles in jar #5*)
- ▷ $9 \leq$ `jar6` ≤ 10 (*weight of marbles in jar #6*)
- ▷ $9 \leq$ `jar7` ≤ 10 (*weight of marbles in jar #7*)
- ▷ $9 \leq$ `jar8` ≤ 10 (*weight of marbles in jar #8*)
- ▷ $9 \leq$ `jar9` ≤ 10 (*weight of marbles in jar #9*)
- ▷ $9 \leq$ `jar10` ≤ 10 (*weight of marbles in jar #10*)

Restrictions

The following restriction(s) is/are imposed on the solution:

- ▷ You cannot use selection statements such as (*but not limited to*) **`if`**, **`switch`**, or `?:` operator
- ▷ You cannot use repetition statements such as (*but not limited to*) **`while`**, **`do-while`**, or **`for`**
- ▷ You cannot use `<`**`math`**`.`**`h`**`>` library

Tasks

The problem is split into 4 task(s). In the sample run, please note the following:

- ↩ is the *invisible* `[`**`newline`**`]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/4

Write a program to read **ten (10)** **`integer`** numbers corresponding to the weight of each marbles in each jar and print these numbers back separated by a single `[`**`space`**`]` and terminated by a `[`**`newline`**`]`.

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 10 9`	`10 10 10 10 10 10 10 10 10 9`↩

Save your program in the file named `ten_jar1.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 2*), copy your program using the following command:

```
cp ten_jar1.c ten_jar2.c
```

Task 2/4

Write a program to read **ten (10)** `integer` numbers corresponding to the weight of each marbles in each jar and print a single `integer` corresponding to the sum of all the numbers in an even-numbered jars terminated by a [`newline`].

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 10 9`	`49↩`

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 9 10`	`50↩`

Save your program in the file named `ten_jar2.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 3*), copy your program using the following command:

```
cp ten_jar2.c ten_jar3.c
```

Task 3/4

Write a program to read **ten (10)** `integer` numbers corresponding to the weight of each marbles in each jar and print a single `integer` corresponding to the sum of all the numbers multiplied by the jar number terminated by a [`newline`].

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 10 9`	`540↩ | 10 + 20 + 30 + 40 + 50 + 60 + 70 + 80 + 90 + 90`

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 9 10`	`541↩ | 10 + 20 + 30 + 40 + 50 + 60 + 70 + 80 + 81 + 100`

Save your program in the file named `ten_jar3.c`. Submit your program to CodeCrunch. To proceed to the next task (*e.g., task 4*), copy your program using the following command:

```
cp ten_jar3.c ten_jar4.c
```

Task 4/4

Write a program to read **ten (10)** `integer` numbers corresponding to the weight of each marbles in each jar and print a single `integer` corresponding to the jar number with lighter marbles.

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 10 9`	`10↩`

Sample Run:

Inputs:	Outputs:
`10 10 10 10 10` `10 10 10 9 10`	`9↩`

Save your program in the file named `ten_jar4.c`. Submit your program to CodeCrunch.