# Abundant

1. Find all abundant numbers between two given inclusive limits (positive). An abundant number is a number for which the sum of its proper divisors (1 included) is greater than the number itself. Group the numbers by the first digit and print them out.

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- IndexOutOfRange exception --> 'index out of range'  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
1 133  
output:  
12 18 100 102 104 108 112 114 120 126 132  
20 24  
30 36  
40 42 48  
54 56  
60 66  
70 72 78  
80 84 88  
90 96

Solutions :  
  
using System;

using System.Collections.Generic;

namespace \_08Abundant

{

internal class Program

{

static void Main(string[] args)

{

try

{

string[] input = Console.ReadLine()?.Split(' ');

if (input == null || input.Length != 2 || !int.TryParse(input[0], out int limit1) || !int.TryParse(input[1], out int limit2))

{

Console.WriteLine("converting problem");

return;

}

int startLimit = Math.Min(limit1, limit2);

int endLimit = Math.Max(limit1, limit2);

// Dictionary to hold abundant numbers grouped by their first digit

Dictionary<char, List<int>> abundantNumbers = new Dictionary<char, List<int>>();

for (int num = startLimit; num <= endLimit; num++)

{

int divisorsSum = 1;

for (int i = 2; i <= Math.Sqrt(num); i++)

{

if (num % i == 0)

{

divisorsSum += i;

if (i != num / i)

{

divisorsSum += num / i;

}

}

}

if (divisorsSum > num)

{

char firstDigit = num.ToString()[0];

if (!abundantNumbers.ContainsKey(firstDigit))

{

abundantNumbers[firstDigit] = new List<int>();

}

abundantNumbers[firstDigit].Add(num);

}

}

// Sort the keys (first digits)

var sortedKeys = new List<char>(abundantNumbers.Keys);

sortedKeys.Sort();

// Print the numbers grouped by their first digit

foreach (var key in sortedKeys)

{

Console.WriteLine(string.Join(" ", abundantNumbers[key]));

}

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("index out of range");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Hits per Year

1. Read in an artist file (download files: [files.zip](https://thomasmore.instructure.com/courses/32557/files/6415235?wrap=1)[Download files.zip](https://thomasmore.instructure.com/courses/32557/files/6415235/download?download_frd=1), files already in CodeGrade)  and print the number of hits the artist had per year. Make your program as bulletproof as possible, otherwise, print 'crazy input'

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- FileNotFound Exception --> 'file not found'  
- Any other exception --> 'crazy input'

**Input/Output**

input:  
Khalid.txt  
output:  
2016: 5 hit(s)  
2017: 17 hit(s)  
2018: 16 hit(s)  
2019: 19 hit(s)  
2020: 3 hit(s)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Threading;

namespace \_08HitsPerYear

{

internal class Program

{

static void Main(string[] args)

{

try

{

string file = Console.ReadLine();

Dictionary<int, int> aggregatedHitsPerYear = new Dictionary<int, int>();

using (StreamReader reader = new StreamReader(file))

{

string line;

while ((line = reader.ReadLine()) != null)

{

string[] parts = line.Split(';');

if (parts.Length >= 5)

{

if (int.TryParse(parts[4].Substring(0, Math.Min(4, parts[4].Length)), out int year))

{

// Increment the hit count for the year

if (!aggregatedHitsPerYear.ContainsKey(year))

aggregatedHitsPerYear[year] = 1;

else

aggregatedHitsPerYear[year]++;

}

}

}

}

foreach (var yearEntry in aggregatedHitsPerYear.OrderBy(entry => entry.Key))

{

int year = yearEntry.Key;

int hits = yearEntry.Value;

Console.WriteLine($"{year}: {hits} hit(s)");

}

}

catch (FileNotFoundException)

{

Console.WriteLine("file not found");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

Explanation :

Imagine we have a file with the following lines:

hit1;data;data;data;2020-01-01

hit2;data;data;data;2019-05-23

hit3;data;data;data;2020-07-15

hit4;data;data;data;2021-03-11

Here's what the code does:

1. Opens the file.
2. Reads the first line: hit1;data;data;data;2020-01-01.
3. Splits it into parts: ["hit1", "data", "data", "data", "2020-01-01"].
4. Checks there are at least 5 parts (true).
5. Takes the fifth part: 2020-01-01.
6. Extracts the year (first four characters): 2020.
7. Converts 2020 to a number and checks if it’s already in the dictionary.
   * It’s not, so adds 2020 with a count of 1.

Repeats for the next lines:

* 2019-05-23 becomes 2019, added with a count of 1.
* 2020-07-15 becomes 2020, count is increased to 2.
* 2021-03-11 becomes 2021, added with a count of 1.

After processing all lines, the dictionary aggregatedHitsPerYear looks like this:

* 2020: 2 hits
* 2019: 1 hit
* 2021: 1 hit

The code then prints:

2019: 1 hit(s)

2020: 2 hit(s)

2021: 1 hit(s)

# Sort

1. Read a list of personal names (<first name> <last name>) and sort them by the last name!

Make your program as bulletproof as possible, otherwise, print 'crazy input'  
- Format exception --> 'converting problem'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
Evy-Rose Lasting, Anne Thurium, Cherry Blossom, Glad Oli, Ginger Plant  
output:  
Cherry Blossom  
Evy-Rose Lasting  
Glad Oli  
Ginger Plant  
Anne Thurium

input:  
Sly Meebuggah, Saul Goodmate, Faye Clether, Sarah Moanees, Ty Ayelloribbin, Hugo First, Olive Tree, Fran Gio Pani, John Quil  
output:  
Ty Ayelloribbin  
Faye Clether  
Hugo First  
Fran Gio Pani  
Saul Goodmate  
Sly Meebuggah  
Sarah Moanees  
John Quil  
Olive Tree

using System;

using System.Collections.Generic;

namespace \_08Sort

{

internal class Program

{

static void Main(string[] args)

{

try

{

string input = Console.ReadLine();

foreach (char c in input)

{

if (char.IsDigit(c))

{

Console.WriteLine("converting problem");

return;

}

}

string[] nameList = input.Split(',');

if (nameList.Length < 2)

{

Console.WriteLine("crazy input");

return;

}

List<string> personalNames = new List<string>();

foreach (var nameWithComma in nameList)

{

string[] nameParts = nameWithComma.Trim().Split(' ');

if (nameParts.Length >= 2)

{

string firstName = nameParts[0];

string lastName = string.Join(" ", nameParts, 1, nameParts.Length - 1);

personalNames.Add($"{firstName} {lastName}");

}

else

{

Console.WriteLine("crazy input");

return;

}

}

personalNames.Sort((name1, name2) =>

{

string[] parts1 = name1.Split(' ');

string[] parts2 = name2.Split(' ');

return StringComparer.OrdinalIgnoreCase.Compare(parts1[1], parts2[1]);

});

foreach (var name in personalNames)

{

Console.WriteLine(name);

}

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Zero

1. Read in a series of integers and find the two elements whose sum is closest to zero. The sum has to be positive. There are no duplicates in the array.

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
-1 -5 2 7 -6  
output:  
1  
-1 2

input:  
12 -7 3 2 -6 5 -9 8 -2 15  
output:  
0  
2 -2

using System;

namespace \_08Zero

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read input

string input = Console.ReadLine();

// Split input into array of strings

string[] stringNumbers = input.Split(' ');

// Check if there are at least 2 numbers

if (stringNumbers.Length < 2)

{

Console.WriteLine("invalid input");

return;

}

// Convert string array to int array

int[] numbers = new int[stringNumbers.Length];

for (int i = 0; i < stringNumbers.Length; i++)

{

if (!int.TryParse(stringNumbers[i], out numbers[i]))

{

Console.WriteLine("converting problem");

return;

}

}

int closestSum = int.MaxValue;

int firstElement = 0;

int secondElement = 0;

// Iterate over the array to find the two elements with the closest sum to zero

for (int i = 0; i < numbers.Length; i++)

{

for (int j = i + 1; j < numbers.Length; j++)

{

int currentSum = numbers[i] + numbers[j];

// Only consider positive sums

if (currentSum > 0 && currentSum < closestSum)

{

closestSum = currentSum;

firstElement = numbers[i];

secondElement = numbers[j];

}

}

}

// Print the closest positive sum and the two elements

Console.WriteLine($"{closestSum}");

Console.WriteLine($"{firstElement} {secondElement}");

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Find

1. Find all duck numbers from a sequence and print them out in sorted order. A duck number is a number that contains a 0 in (but doesn't start with a 0!).

Make your program as bulletproof as possible, otherwise, print 'crazy input'.

**Input/Output**  
input:  
10,11,20,1.09,100  
output:  
1.09 10 20 100

input:  
happy,sad,this is a 0,0.9, 10.9  
output:  
10.9

using System;

using System.Collections.Generic;

namespace \_08Find

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read input

string inputSequence = Console.ReadLine();

string[] elements = inputSequence.Split(',');

List<decimal> duckNumbers = new List<decimal>();

foreach (string element in elements)

{

string trimmedElement = element.Trim();

if (decimal.TryParse(trimmedElement, out decimal number))

{

string numberStr = trimmedElement;

// Check if the number is a duck number

if (numberStr.Contains('0') && numberStr[0] != '0')

{

duckNumbers.Add(number);

}

}

}

// Sort duck numbers

duckNumbers.Sort();

// Print duck numbers

if (duckNumbers.Count > 0)

{

Console.WriteLine(string.Join(" ", duckNumbers));

}

else

{

Console.WriteLine("No duck elements found.");

}

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Out with odd

1. Write a program to remove from a sequence all the integers, which appear an odd number of times.

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**input  
4 2 2 5 2 3 2 3 1 5 2 6 6 6  
output  
5 3 3 5

input  
3 5 1 7 6 6 2 1 3 5 3 1 3  
output  
3 5 6 6 3 5 3 3

using System;

using System.Collections.Generic;

namespace \_08OutWithOdd

{

internal class Program

{

static void Main(string[] args)

{

try

{

string inputSequence = Console.ReadLine();

string[] stringNumbers = inputSequence.Split(' ');

List<int> numbers = new List<int>();

// Convert input strings to integers

foreach (var str in stringNumbers)

{

if (int.TryParse(str, out int num))

{

numbers.Add(num);

}

else

{

Console.WriteLine("converting problem");

return;

}

}

Dictionary<int, int> countDict = new Dictionary<int, int>();

// Count occurrences of each number

foreach (int num in numbers)

{

if (countDict.ContainsKey(num))

{

countDict[num]++;

}

else

{

countDict[num] = 1;

}

}

List<int> result = new List<int>();

// Filter out numbers with odd occurrences

foreach (int num in numbers)

{

if (countDict[num] % 2 == 0)

{

result.Add(num);

}

}

// Print the result as a space-separated string

if (result.Count > 0)

{

Console.WriteLine(string.Join(" ", result));

}

else

{

Console.WriteLine("No elements with even occurrences found.");

}

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Fill up the dictionary

1. Read in a key-value pair. Both are integers and the keys always go from 1 to 9. There's one key-value pair missing. Print out the missing key and the value based on the pattern of integers provided in the other pairs.

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
1-5 2-10 3-15 5-25 6-30 7-35 8-40 9-45  
output:  
4-20

input:  
1-7 3-21 4-28 5-35 6-42 7-49 8-56 9-63  
output:  
2-14

using System;

namespace \_08FillupDictionary

{

internal class Program

{

static void Main(string[] args)

{

try

{

string input = Console.ReadLine();

if (string.IsNullOrWhiteSpace(input))

{

Console.WriteLine("crazy input");

return;

}

string[] pairs = input.Split(' ');

int[] values = new int[9];

foreach (string pair in pairs)

{

string[] parts = pair.Split('-');

if (parts.Length != 2)

{

Console.WriteLine("converting problem");

return;

}

if (int.TryParse(parts[0], out int key) && int.TryParse(parts[1], out int value))

{

if (key < 1 || key > 9)

{

Console.WriteLine("crazy input");

return;

}

if (values[key - 1] != 0)

{

Console.WriteLine("crazy input");

return;

}

values[key - 1] = value;

}

else

{

Console.WriteLine("converting problem");

return;

}

}

int missingKey = 0;

int missingValue = 0;

for (int i = 1; i <= 9; i++)

{

if (values[i - 1] == 0)

{

missingKey = i;

if (i == 1)

{

missingValue = 2 \* values[1] - values[2];

}

else if (i == 9)

{

missingValue = 2 \* values[7] - values[6];

}

else

{

missingValue = (values[i - 2] + values[i]) / 2;

}

break;

}

}

if (missingKey == 0)

{

Console.WriteLine("crazy input");

return;

}

Console.WriteLine($"{missingKey}-{missingValue}");

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Emordnilap

An emordnilap is a word that forms another valid word when its letters are reversed. For example:

* The word "reward" reversed is "drawer."
* The word "stressed" reversed is "desserts."

In simple terms, an emordnilap is like a palindrome but results in a different word when reversed.

1. An emordnilap is a word that results in a different existing word when it is reversed! Read in a sentence and print out a sorted distinct (no duplicates!) list of all emordnilap-words! Check the reverse word through the English word file ([english.txt](https://thomasmore.instructure.com/courses/32557/files/6415254?wrap=1)[Download english.txt](https://thomasmore.instructure.com/courses/32557/files/6415254/download?download_frd=1)).

Make your program as bulletproof as possible, otherwise, print 'crazy input'  
- Format exception --> 'converting problem'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
The reward of a work is to have produced it.  
output:  
is it of reward

input:  
Be the person your dog thinks you are.  
output:  
are be dog

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace \_08\_Emordnilap

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read and process the input sentence

string input = Console.ReadLine();

if (string.IsNullOrWhiteSpace(input))

{

Console.WriteLine("crazy input");

return;

}

string[] inputSentence = input.Trim('.').Split();

List<string> emordnilap = new List<string>();

// Path to the dictionary file

string cwd = Directory.GetCurrentDirectory();

string path = Path.Combine(cwd, "english.txt");

// Read the dictionary file

List<string> dictionary = File.ReadAllLines(path).Select(line => line.Trim().ToLower()).ToList();

// Iterate through each word in the input sentence

foreach (string word in inputSentence)

{

// Reverse the word

char[] chars = word.ToCharArray();

Array.Reverse(chars);

string reversedWord = new string(chars).ToLower();

// Check if the reversed word is in the dictionary and is different from the original word

if (dictionary.Contains(reversedWord) && !reversedWord.Equals(word.ToLower()))

{

if (!emordnilap.Contains(word.ToLower()))

{

emordnilap.Add(word.ToLower());

}

}

}

// Sort the list of emordnilap words

emordnilap.Sort();

// Display the sorted list

Console.WriteLine(string.Join(" ", emordnilap));

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Most used smallest word

1. Read in a file (download files: [Boeken.zip](https://thomasmore.instructure.com/courses/32557/files/6415250?wrap=1)[Download Boeken.zip](https://thomasmore.instructure.com/courses/32557/files/6415250/download?download_frd=1), files already in CodeGrade)  and find the most used smallest word (at least 2 characters).

Make your program as bulletproof as possible, otherwise, print 'crazy input'  
- FileNotFound exception --> 'file not found'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
DeVolmaakte.txt  
output:  
de

input:  
LoremIpsum.txt  
output:  
in

using System;

using System.Collections.Generic;

using System.IO;

namespace \_08mostUsed

{

internal class Program

{

static void Main(string[] args)

{

try

{

string filePath = Console.ReadLine();

if (string.IsNullOrEmpty(filePath))

{

Console.WriteLine("crazy input");

return;

}

string[] words;

try

{

words = File.ReadAllText(filePath)

.Split(new char[] { ' ', '\t', '\n', '\r', '.', ',', ';', ':', '!', '?' },

StringSplitOptions.RemoveEmptyEntries);

}

catch (FileNotFoundException)

{

Console.WriteLine("file not found");

return;

}

catch (Exception)

{

Console.WriteLine("crazy input");

return;

}

Dictionary<string, int> wordCounts = new Dictionary<string, int>();

foreach (var word in words)

{

string lowerWord = word.ToLower();

if (lowerWord.Length >= 2)

{

if (!wordCounts.ContainsKey(lowerWord))

{

wordCounts[lowerWord] = 0;

}

wordCounts[lowerWord]++;

}

}

string mostUsedSmallestWord = null;

foreach (var kvp in wordCounts)

{

if (mostUsedSmallestWord == null ||

kvp.Key.Length < mostUsedSmallestWord.Length ||

(kvp.Key.Length == mostUsedSmallestWord.Length && kvp.Value > wordCounts[mostUsedSmallestWord]))

{

mostUsedSmallestWord = kvp.Key;

}

}

if (mostUsedSmallestWord != null)

{

Console.WriteLine(mostUsedSmallestWord);

}

else

{

Console.WriteLine("crazy input");

}

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Least used longest word

1. Read in a file (download files: [Boeken.zip](https://thomasmore.instructure.com/courses/32557/files/6415250?wrap=1)[Download Boeken.zip](https://thomasmore.instructure.com/courses/32557/files/6415250/download?download_frd=1), files already in CodeGrade)  and find the least used longest word.

Make your program as bulletproof as possible, otherwise, print 'crazy input'  
- FileNotFound exception --> 'file not found'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
DeRoos.txt  
output:  
heuvelachtigheid

input:  
DeStilleKracht.txt  
output:  
tusschenruimten

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace \_08Leastusedlongestword

{

internal class Program

{

static void Main(string[] args)

{

try

{

string filename = Console.ReadLine();

if (!File.Exists(filename))

{

Console.WriteLine("file not found");

return;

}

string text = File.ReadAllText(filename);

string[] words = text.Split(new char[] { ' ', '\t', '\n', '\r', '.', ',', ';', ':', '-', '!', '?', '(', ')' }, StringSplitOptions.RemoveEmptyEntries);

var wordCounts = new Dictionary<string, int>();

foreach (string word in words)

{

string lowerWord = word.ToLower(); // Convert to lower case for case insensitive comparison

if (lowerWord.Length >= 2)

{

if (wordCounts.ContainsKey(lowerWord))

{

wordCounts[lowerWord]++;

}

else

{

wordCounts[lowerWord] = 1;

}

}

}

var longestWords = wordCounts.Where(pair => pair.Key.Length == wordCounts.Keys.Max(key => key.Length)).ToList();

if (longestWords.Count == 0)

{

Console.WriteLine("crazy input");

return;

}

var leastUsedWord = longestWords.OrderBy(pair => pair.Value).First();

Console.WriteLine(leastUsedWord.Key);

}

catch (FileNotFoundException)

{

Console.WriteLine("file not found");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

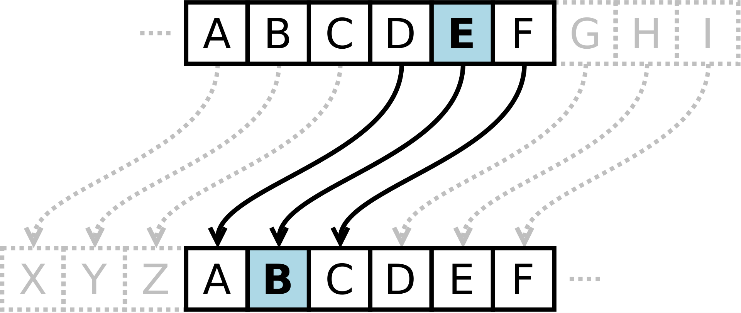
}

}

# Caesar

1. Protect a given message by a simple cryptographic rule!

Caesar shift is one of the most straightforward and widely known encryption techniques. It is a substitution cipher in which each uppercase letter is replaced by another uppercase letter with a fixed number of positions up or down the alphabet. For example, with a shift of -3, D would be replaced by A, E would become B, and so on.



Read in a number (how many places to shift) and a word (only consisting of alphabetical characters) and output the encrypted word.

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
2  
sugar  
output:  
UWICT

input:  
3  
SUGAR 321  
output:  
crazy input

using System;

namespace CaesarCipher

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read shift value

string shiftInput = Console.ReadLine();

if (!int.TryParse(shiftInput, out int shift))

{

Console.WriteLine("converting problem");

return;

}

int shift1 = shift % 26;

// Read word

string word = Console.ReadLine();

if (string.IsNullOrEmpty(word) || !IsAlphabetical(word))

{

Console.WriteLine("crazy input");

return;

}

// Encrypt the word

string encryptedWord = EncryptCaesarCipher(word, shift1);

// Print the encrypted word

Console.WriteLine(encryptedWord);

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

// Method to check if a string contains only alphabetical characters

private static bool IsAlphabetical(string input)

{

foreach (char c in input)

{

if (!char.IsLetter(c))

{

return false;

}

}

return true;

}

// Method to encrypt a word using Caesar Cipher

private static string EncryptCaesarCipher(string word, int shift)

{

char[] buffer = word.ToUpper().ToCharArray();

for (int i = 0; i < buffer.Length; i++)

{

char letter = buffer[i];

letter = (char)(letter + shift);

if (letter > 'Z')

{

letter = (char)(letter - 26);

}

else if (letter < 'A')

{

letter = (char)(letter + 26);

}

buffer[i] = letter;

}

return new string(buffer);

}

}

}

# Sum

1. Read in an integer N and find the sum of all squared integers between 1 and N (exclusive).  
     
   Make your program as bulletproof as possible and print out clear error messages, according to this list:  
   - Format exception --> 'converting problem  
   - Overflow exception --> 'too big for this type'  
   - Any other exception --> 'crazy input'

**Input/Output**  
input:  
85  
output:  
201110

input:  
3.2  
output:  
converting problem

using System;

namespace \_07Sum

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read and parse the input

if (!int.TryParse(Console.ReadLine(), out int N))

{

throw new FormatException("converting problem");

}

long sum = 0;

int endValue = Math.Abs(N); // Handle negative input

// Calculate the sum of squares

for (int i = 1; i < endValue; i++)

{

checked

{

sum += (long)i \* i;

}

}

// Output the result

Console.WriteLine($"{sum}");

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Product

13 . Read in a positive integer N and find the minimal positive integer Q so that the product of the digits of Q is equal to N.  
E.g.: If I want to get 21 (like the first input) I have to find the lowest possible positive number in which I multiply the digits to get 21 --> in this case 37, because 3 \* 7 = 21

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
17.8  
output:  
converting problem

input:  
42  
output:  
67

using System;

namespace \_07Product

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read input and validate it

if (!int.TryParse(Console.ReadLine(), out int N) || N <= 0)

{

throw new FormatException("converting problem");

}

// Helper function to calculate the product of the digits of a number

int calculateProduct(int num)

{

int product = 1;

while (num > 0)

{

int digit = num % 10;

product \*= digit;

num /= 10;

}

return product;

}

// Find the minimal positive integer Q

for (int Q = 1; ; Q++)

{

if (calculateProduct(Q) == N)

{

Console.WriteLine($"{Q}");

return;

}

}

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Run-length encoding

1. Run-length encoding is the replacement of repeating patterns of characters with the number of repetitions plus the character.  
   For example, the string AaaabbbdddCCCCeeeeeffff can be compressed by A3a3b3d4C5e4f (we only consider alphabetical characters for this compression method)

Make your program as bulletproof as possible and print out clear error messages, according to this list:   
- Format exception --> 'converting problem  
- Overflow exception --> 'too big for this type'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
321456a  
output:  
crazy input

input:  
aaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaa  
output:  
26a32b4a

using System;

using System.Text;

namespace \_07RunLengthEncoding

{

internal class Program

{

static void Main(string[] args)

{

try

{

string inputStr = Console.ReadLine().Trim();

if (string.IsNullOrEmpty(inputStr))

{

throw new Exception("crazy input");

}

StringBuilder result = new StringBuilder();

int i = 0;

while (i < inputStr.Length)

{

char currentChar = inputStr[i];

if (!char.IsLetter(currentChar))

{

throw new Exception("crazy input");

}

int count = 1;

while (i + 1 < inputStr.Length && inputStr[i] == inputStr[i + 1])

{

i++;

count++;

}

if (count > 1)

{

result.Append(count);

}

result.Append(currentChar);

i++;

}

Console.WriteLine(result.ToString());

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Sator Square

14. Check if a given 5-by-5 matrix is a sator square.  
The SATOR square is a 5-by-5 matrix containing 5 words that indicate the same thing in all directions (left to right, right to left, top to bottom, bottom to top).

🡪 

**It should work for both capital and regular letters.**

If it does match up you print out: "This is a correct satorsquare", otherwise "This is NOT a correct satorsquare"

Make your program as bulletproof as possible and print out clear error messages, according to this list:

-If it contains a number, you forward it to a FormatException.   
-If any of the arrays in the matrix is longer or shorter than 5, you forward it to an IndexOutOfRangeException.  
- Format exception --> 'you have entered a number or the input is wrong'  
- IndexOutOfRangeException --> 'index out of range or problem with array length'  
- Any other exception --> 'crazy input'

TIP: use 'throw new ...();' Instead of the ... you can fill in any type of exception  
e.g. When you come across a number in the string you've input. The computer thinks it's part of the string, but we don't want to have a number there.)

**Input/Output**  
input:  
R O T a s  
O p e R A  
T E N E T  
a r E P O  
s a t O R  
output:  
This is a correct satorsquare

input:  
L E B E N  
e h e r e  
b 2 I e 7  
9 R e h E  
N e b E L  
output:  
you have entered a number or the input is wrong

using System;

using System.Linq;

class Program

{

public static bool IsSatorSquare(char[,] tablet)

{

int n = tablet.GetLength(0);

// Check for all Sator Square conditions

return !Enumerable.Range(0, n).Any(i =>

Enumerable.Range(0, n).Any(j =>

tablet[i, j] != tablet[j, i] ||

tablet[i, j] != tablet[n - i - 1, n - j - 1]

)

);

}

static void Main()

{

try

{

char[,] matrix = new char[5, 5];

for (int i = 0; i < 5; i++)

{

string inputRow = Console.ReadLine();

// Remove spaces and convert to uppercase

inputRow = inputRow.Replace(" ", "").ToUpper();

// Check for proper length and digits before processing

if (inputRow.Length != 5 || inputRow.Any(char.IsDigit))

{

throw new FormatException();

}

for (int j = 0; j < 5; j++)

{

matrix[i, j] = inputRow[j];

}

}

bool isSatorSquare = IsSatorSquare(matrix);

if (isSatorSquare)

{

Console.WriteLine("This is a correct satorsquare");

}

else

{

Console.WriteLine("This is NOT a correct satorsquare");

}

}

catch (FormatException)

{

Console.WriteLine("you have entered a number or the input is wrong");

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("index out of range or problem with array length");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

# Selective sum

* 1. Read in an amount of integers, split by a space. Next, give a starting point and an endpoint.  
     The numbers in between (including the start and endpoint) need to be added together and printed out.

Example:  
10 20 30 45 51  
2  
4  
---------------------  
95 --> 20+30+45  
The number 2 indicates the second number and the number 4 the 4th number. So keep this in account when finding your index.

Make your program as bulletproof as possible and print out clear error messages, according to this list:

- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- IndexOutOfRangeException --> 'index out of range'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
7 28 39 17 2 5 9  
5  
7  
output:  
16

input:  
3 9 2 1.7 8 5 4  
1  
5  
output:  
converting problem

using System;

namespace \_07SelectiveSum

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read input numbers and split them by space

string inputNumbers = Console.ReadLine();

string[] numberStrings = inputNumbers.Split(' ');

// Convert number strings to integers

int[] numbers = new int[numberStrings.Length];

for (int i = 0; i < numberStrings.Length; i++)

{

if (!int.TryParse(numberStrings[i], out numbers[i]))

{

throw new FormatException("converting problem");

}

}

// Read start and end indices

if (!int.TryParse(Console.ReadLine(), out int startIndex))

{

throw new FormatException("converting problem");

}

if (!int.TryParse(Console.ReadLine(), out int endIndex))

{

throw new FormatException("converting problem");

}

// Check if indices are within valid range

if (startIndex < 1 || endIndex > numbers.Length || startIndex > endIndex)

{

throw new IndexOutOfRangeException("index out of range");

}

// Calculate the sum of the numbers between the start and end indices (inclusive)

int sum = 0;

for (int i = startIndex - 1; i < endIndex; i++)

{

checked

{

sum += numbers[i];

}

}

// Print the result

Console.WriteLine($"{sum}");

}

catch (FormatException ex)

{

Console.WriteLine(ex.Message);

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (IndexOutOfRangeException ex)

{

Console.WriteLine(ex.Message);

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

The checked keyword in C# is used to explicitly enable overflow checking for arithmetic operations and conversions. When an arithmetic operation exceeds the range of the data type, an OverflowException is thrown. This helps prevent silent overflow, where values wrap around without any error.

# Words

* 1. Select words consisting only of letters in a certain file (already in CodeGrade [files.zip](https://thomasmore.instructure.com/courses/32557/files/6415235?wrap=1)[Download files.zip](https://thomasmore.instructure.com/courses/32557/files/6415235/download?download_frd=1)) of a certain length that also contain specific characters. Read the file first, then the characters, and finally the number of characters. Print each unique word that meets the conditions.

Make your program as bulletproof as possible and print out clear error messages, according to this list:  
- Format exception --> 'converting problem'  
- Overflow exception --> 'too big for this type'  
- FileNotFound exception --> 'file not found'  
- Any other exception --> 'crazy input'

**Input/Output**  
input:  
ColdPlay.txt  
the  
5  
output:  
heart death there teeth those other their three earth chest white these theme neath threw theon thief

input:  
CardiC.txt  
az  
7  
output:  
file not found

using System;

using System.IO;

using System.Linq;

namespace \_07RunLengthencoding

{

internal class Program

{

static void Main(string[] args)

{

try

{

// Read file name

string fileName = Console.ReadLine();

// Read characters

string characters = Console.ReadLine();

// Read and parse length

if (!int.TryParse(Console.ReadLine(), out int length))

{

throw new FormatException();

}

// Read the file content

string fileContent = File.ReadAllText(fileName);

// Split the content into words using specified delimiters

string[] allWords = fileContent.Split(new[]

{

' ', '\t', '\n', '\r', '\f', '\v', '.', ',', ';', ':', '-', '!', '?', '(', ')', '[', ']', '{', '}', '"', '\''

}, StringSplitOptions.RemoveEmptyEntries);

// Filter words based on the specified conditions

var filteredWords = allWords

.Where(word => word.Length == length && characters.All(word.Contains))

.Distinct(StringComparer.OrdinalIgnoreCase)

.OrderBy(word => word) // Optional: Sort the words alphabetically

.ToArray();

// Print the result

Console.WriteLine(string.Join(" ", filteredWords).ToLower());

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (FileNotFoundException)

{

Console.WriteLine("file not found");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}

}

# Ending

* 1. Read in two strings (separated by a space) and find the longest common ending between two given strings.  
       
     Note: use of Linq-library or Array-library is forbidden!

**Input/Output**  
input:  
dream stream  
output:  
ream

input:  
cornflakes kellogs  
output:  
s

using System;

class Program

{

static void Main()

{

// Read input from the user

Console.WriteLine("Enter two strings separated by a space:");

string input = Console.ReadLine();

// Split the input into two strings

string[] parts = input.Split(' ');

string firstString = parts[0];

string secondString = parts[1];

// Get the lengths of both strings

int len1 = firstString.Length;

int len2 = secondString.Length;

// Initialize pointers to the end of each string

int index1 = len1 - 1;

int index2 = len2 - 1;

// Move backwards from the end and find the longest common ending

while (index1 >= 0 && index2 >= 0 && firstString[index1] == secondString[index2])

{

index1--;

index2--;

}

// Extract the longest common ending

string longestEnding = firstString.Substring(index1 + 1);

// Print the result

Console.WriteLine("Longest common ending: " + longestEnding);

}

}

# Missing

* 1. Read in an array of integers and find the missing number in that array.

Note: use of Linq-library or Array-library is forbidden!

**Input/Output**  
input:  
9 3 6 1 7 5 2 4 8  
output:  
There is no missing link!

input:  
7 2 9 1 6 5 8 3  
output:  
4

using System;

class Program

{

static void Main()

{

Console.WriteLine("Enter integers separated by space:");

string input = Console.ReadLine();

try

{

// Split the input string into individual numbers

string[] numbersStr = input.Split(' ');

int[] numbers = new int[numbersStr.Length];

// Convert the string array to an integer array

for (int i = 0; i < numbersStr.Length; i++)

{

numbers[i] = int.Parse(numbersStr[i]);

}

// Calculate the expected sum of the first n natural numbers

int n = numbers.Length + 1;

int expectedSum = (n \* (n + 1)) / 2;

// Calculate the actual sum of the given numbers

int actualSum = 0;

for (int i = 0; i < numbers.Length; i++)

{

actualSum += numbers[i];

}

// Find the missing number

int missingNumber = expectedSum - actualSum;

// Print the result

if (missingNumber > 0 && missingNumber <= n)

{

Console.WriteLine("Missing number: " + missingNumber);

}

else

{

Console.WriteLine("There is no missing link!");

}

}

catch (FormatException)

{

Console.WriteLine("converting problem");

}

catch (OverflowException)

{

Console.WriteLine("too big for this type");

}

catch (Exception)

{

Console.WriteLine("crazy input");

}

}

}