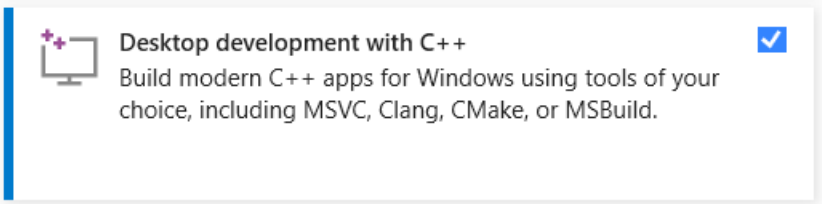
Individual Coursework C1 W1

This coursework is work 30% of the module mark. It is a set of programming exercises designed to assess your ability to apply programming concepts to solve computational problems.

Software Required:

- Visual Studio 2022 for Windows with the "Desktop development with C++" workload installed.
- GitHub Desktop



If you are using a Mac or Linux, then you are advised to run a Windows virtual machine or remote desktop.

In each case, you are building a console application than can be run from Visual Studio or a terminal.

- Each task has a starter project as part of a solution. You must use this.
- For each task, there are specific functions and files you may modify. These are specified for each task in the sections that follow, and you must adhere to this. If you do not, you may lose marks or even score zero.
- Each task also has a set of basic "unit tests" you can run to check your solutions. These provide a good indication if your solution is correct but be aware other tests might be added by the module team during the assessment process.

One of the most common errors made by students is to not read the tasks fully. Please make sure you do this. Also, take note of any comments in the starter code.

Starter Code

You must use this, and you must clone it from GitHub classroom using the following link:

https://classroom.github.com/a/3Qq0t_K3

Select your name - email address and follow the instructions. This should create a private repository that only you and the module staff can access. Clone this repository and **open the solution file in the tasks folder**. You should also save your work in this repository.

Submitting your Work

For your actual submission, you will still need to upload your work to the DLE.

- In Visual Studio, click the build menu and select "clean solution"
- Close Visual Studio
- Delete the hidden `.vs` folder in your tasks folder (it is very large!)
- Archive your task folder as a zip file (should only be a few KB)
- Submit the zip file to the DLE
- Complete the file tasks.docx (included in the repository) and also submit it to the DLE.

Assessment Criteria

The work is marked purely on technical criteria. Several unit tests have been provided to help you check your solutions. Others may be added if deemed necessary.

Source code will be inspected for the following purposes:

- To confirm the work does not break the University rules on plagiarism.
- To confirm the code it written clearly, with appropriate indentation, variable names, and commenting. Up to 10% of the mark may be deducted if code it poorly written.
- To confirm that code has only been edited in the areas permitted.

These tasks are very categorical, so it is quite possible to score 100% on this smaller element. However, be aware it is only worth 30% of the full module mark.

Task 1 – Flow Control

This first task is intended to check you can accurately write logic using if-else if-else statements. The flowchart in Figure 2 shows a flowchart for the problem. You need to follow it precisely.

- Open the solution file in the tasks folder.
- Make task1 the startup project.
- In `task1-functions.cpp`, read the comments and **complete the function** `int setEngineMode(int S, int B)` to implement the logic in Figure 2.

  Do not modify any other function or file.

The inputs parameters are S (speed) and B (battery level). The valid ranges for these are:

$$0 \leq S \leq 100mph$$

$$0\% \leq B \leq 100\%$$

The value return indicates whether the vehicle should run on batteries (0) or petrol (1).

To try different values of `S` and `B`, change the command line properties as follows:

- Right click the task 1 project
- Click Properties
- Select Debugging
- Change the Command Arguments. The first value is the speed (S) and the second is battery level (B). See Figure 1

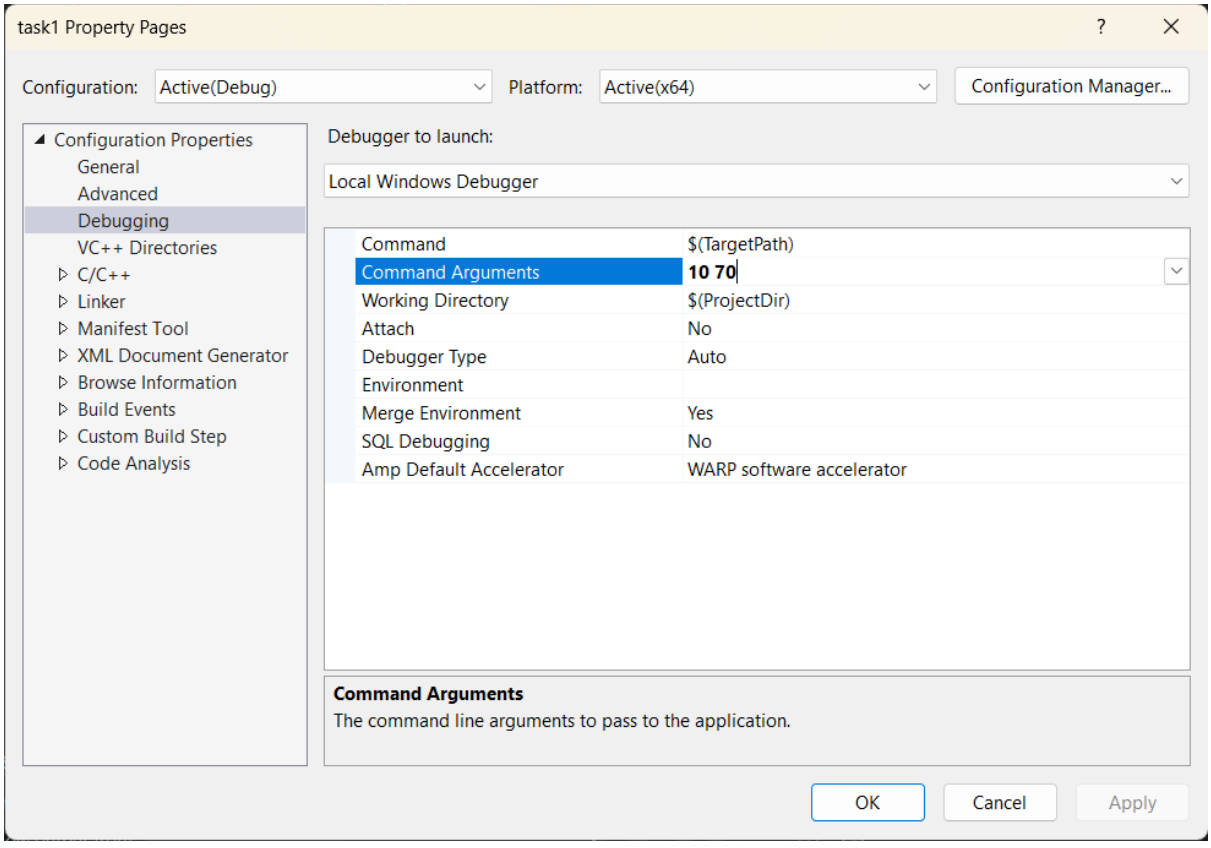


Figure 1. Changing the command line arguments to test different values of speed and battery level.

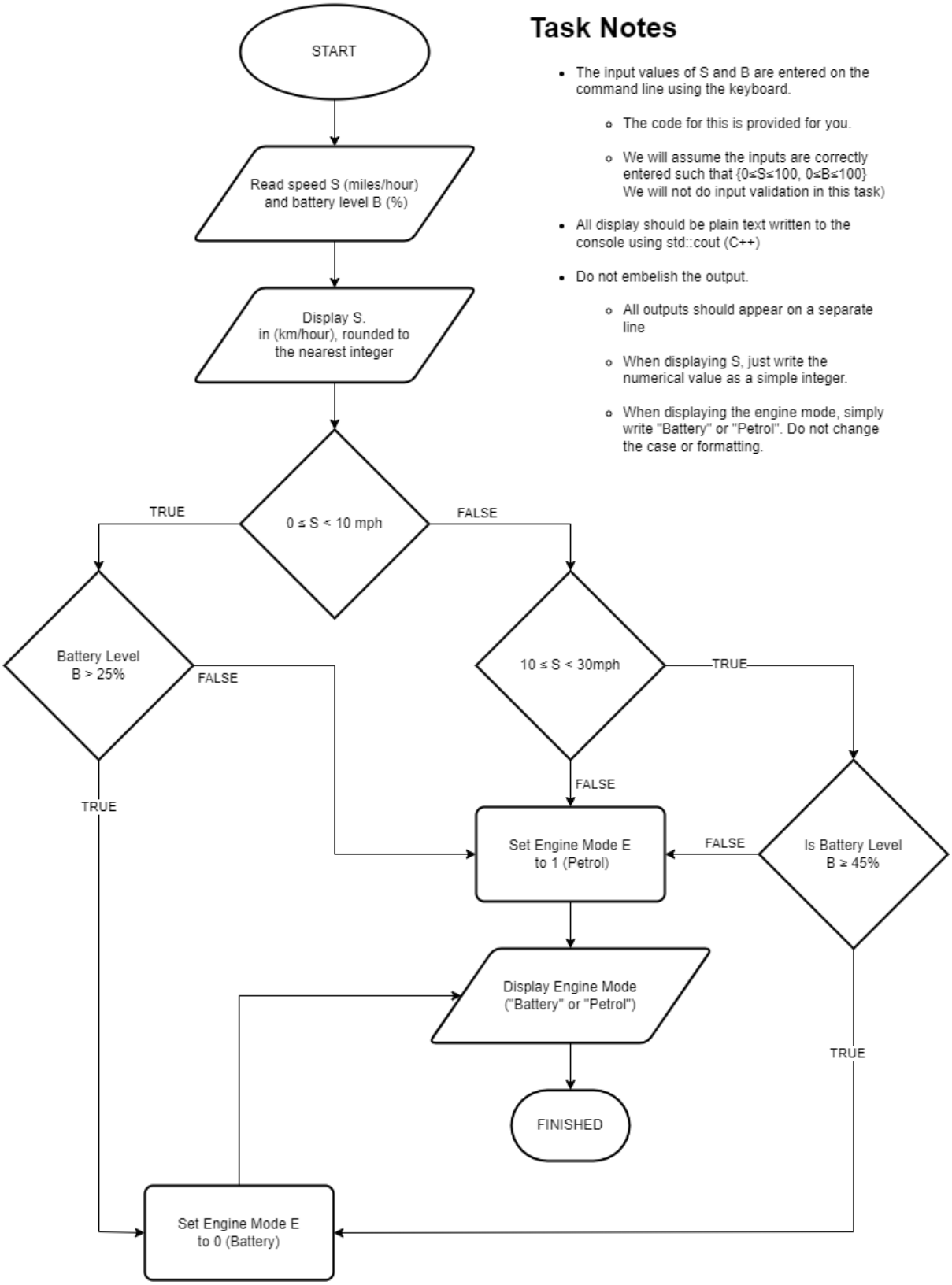


Figure 2. Flow chart for task 1. The speed S and battery level B are assessed to decide whether a car should run on Petrol or Electric power

Task 2 – Calculating properties of a 2D polygon

For this task, you to complete an application that reads coordinates of a 2D shape from a file, calculates the area and circumference, and returns the results.

You are given a partially completed program (including the maths!). You are to modify the following files:

`task2-functions.cpp`

The functions needed have already been created, but they are not finished. You must only complete the functions within these files. You may only modify this in the areas indicated in the comments.

`task2-functions.h`

This contains the declaration of a structure `Point` and some function declarations. You may only modify this in the areas indicated in the comments.

You must NOT change any other files. The `main` function is already complete and must not be changed.

Input File Format

The files you are to read in contain coordinates of a 2D shape. For example, consider the shape in Figure 3.

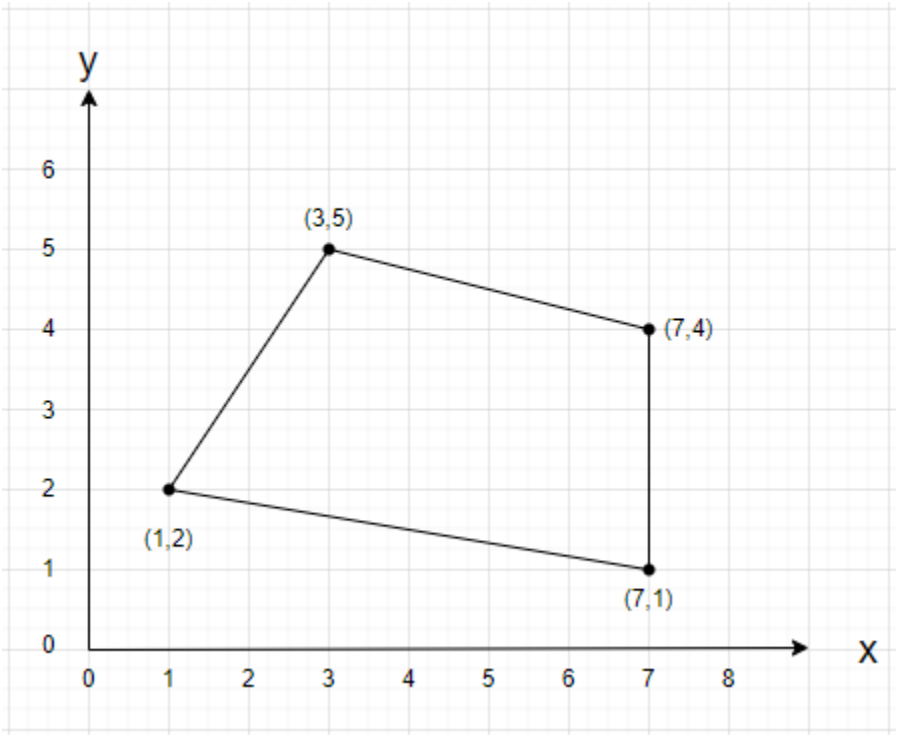


Figure 3. Showing the coordinates of a 2D polygon

The coordinates of each point are stored in a text file. The file contents for this polygon would be formatted as follows:

```
{
    1 2
    3 5
    7 4
    7 1
}
```

Note the opening and closing brackets {} marking the start and end of the data. Any information outside of the braces should be ignored. Whitespace is also ignored.

Setting the data filename

In the task2 folder, two example files are provided for test purposes:

`task2data-1.txt`

This contains the coordinates for a rectangle with area 200 and circumference 60

`task2data-2txt`

This contains the coordinates for a triangle with area 100 and circumference 51.2 (to the nearest single digit).

You can set the filename of the file you wish to read by editing the properties of the task2 project:

- In the Solution Explorer of Visual Studio, right click the task 2 project and click Properties.
- Under Configuration Properties->Debugging->Command Arguments input the filename of the file you want to read (as shown in Figure 4)

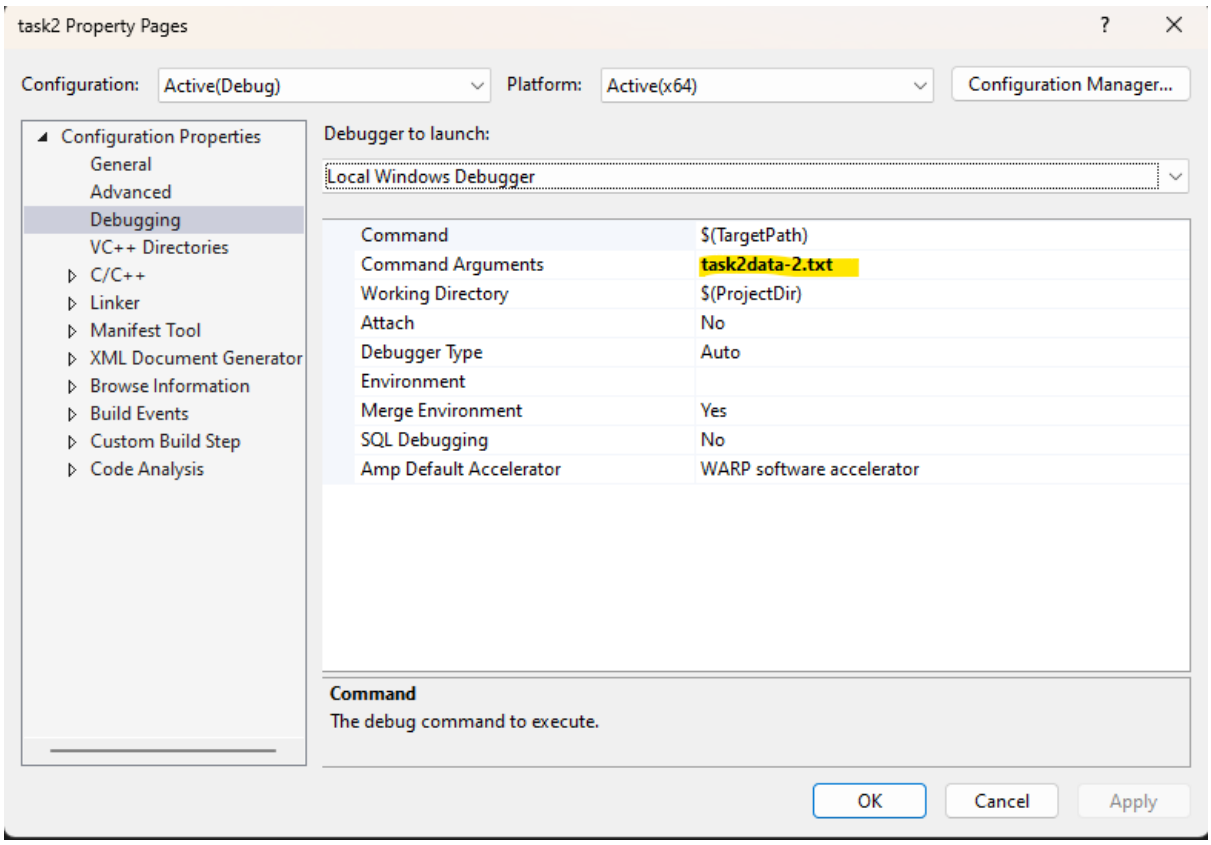


Figure 4. Showing how to set the filename of the input file for task2

Provided Functions

You should ONLY modify the following files:

`task2-functions.h`

`task2-functions.cpp`

Within `task2-functions.h`, you will find the following data type already declared:

```
//The data type for a point - do NOT change this
struct Point {
    double x;
    double y;
};
```

You must use this data type to store the coordinate of each corner in the polygon. Within `task2-functions.cpp`, you will find **the following functions are already provided and completed**:

`static double areaOfPolygon_v1(Point points[], int n)`
Accepts an **array** `points` of type `Point` and the number of points `n`. This contains the mathematics to calculate the area.

`static double distanceBetweenPoints(Point p1, Point p2)`
Accepts a pair of coordinates of type `Point`, calculates the distance between them, and returns the result. You can use this when calculating the circumferance of the polygon.

`void displayPoint(Point p)`
This is a utility function to display a point on the terminal. This is already called from `main`.

Task

Your task is to complete the following functions:

`double areaOfPolygon(vector<Point> points)`
Accepts a **vector** of points (of type `Point`) and returns the area[1].

`double circumferenceOfPolygon(vector<Point> points)`

Accepts a **vector** of points (of type `Point`) and returns the circumference.

`vector<Point> readDataPoints(string strFile)`
Accepts a file path of an input file, reads the file, and returns a **vector** of all the points contained in the file.

Only change the code body in these functions. Do NOT change their name, return type or parameters. Some unit tests[2] are provided to test these functions.

---

[1] vector is used to make this easier, and will be explained in the briefing (with examples)
[2] This will be explained in the briefing.

Expected Output

When you run the project, the output will call the functions you have modified and will attempt to display the following:

- The number of points in the file
- The coordinates of each point (on separate lines)
- The area
- The circumference

You can see the code to do this in the `main` function. For example, if you read the file `task2data-2.txt`, the output should be as follows:

```
3
(0.0,0.0)
(5.0,20.0)
(10.0,0.0)
100.0
51.2
```

The output is already written to the terminal within the `main` function. **Do not change `main`**.

Some examples of input files are provided:

`task2data-1.txt`

`task2data-2.txt`

You should try creating your own input files and check your output. You may use the unit tests provided to help check your answers.

Task 3 – File IO and Data Processing

In this task, you are to write an algorithm to open a file, read its contents into memory, and perform some analysis on the data contained within it.

The data is an NxM grid of zeros (N rows, M columns). Within this grid, there will be a PxQ grid of stars (*), where 1≤P≤N and 1≤Q≤M. All the characters are separated with spaces.

An example is illustrated below in Figure 5. The task is to find the **first rectangle** made of * characters, then find the coordinates of the top left and bottom right of this rectangle. All characters are separated by spaces. Once the first rectangle is found, any further data should be ignored.

To find the first corner:

- Start with row 0 and scan across the columns until you find a * or hit the end of the row.
  - If you find a *, that marks the top left corner.
  - If you don't find a * and reach the end of the row, move down to the next row.
- Repeat until you find it.

The total numbers of rows and characters will vary. Note that it is possible that P=N and Q=M.

		COL								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ROW	**0**	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0
	3	0	0	*	*	*	*	0	0	0
	4	0	0	*	*	*	*	0	0	0
	5	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0

Figure 5. Example file content, with coordinates (3,0) and (4,5). The area in the black box should represents the data in the file. You will need to calculate the number of rows and columns.

For the example in Figure 5, the output of the program would be:

```
{
   "topLeft" : [3,2],
   "bottomRight" : [4,5]
}
```

Note the numbers in square brackets are in the order [row, column]. No other characters or embellishments should be added.

- Open the solution file.
- Make task3 the startup project.
- In `task3-functions.cpp`, complete the function.
  string findCorners (string strFileName)
  Do not modify any other function or file.
- The returned string is displayed in the terminal in main. **Do NOT modify main**.
- Make sure the format of the returned string precisely matches the above (it is case sensitive; don't forget the comma). You can ignore white spaces.

To try different input files, change the command line properties as follows:

- Right click the task 3 project
- Click Properties
- Select Debugging
- Change the Command Arguments. The only value is the name of the file to be read and analysed.

Some examples are already provided (see next). You should also create your own.

Example Files

Some examples of input files have been provided. The first two (and their expected outputs) are shown below:

task3data-1.txt	task3data-2.txt
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 * * * * 0 0 0 0 0 0 0 0 0 0 0 * * * * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 * * 0 0 0 0 * * 0 0 0 0 * * 0 0 0 0 * * 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The expected result would be:
{
"topLeft" : [1,3],
"bottomRight" : [2,6]
}
	The expected result would be:
	{
	"topLeft" : [2,1],
	"bottomRight" : [5,2]
	}

You may use the unit tests provided to help check your answers for other examples.