THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

2021 SCHOLARSHIP EXAMINATION

DEPARTMENT	Computer Science
COURSE TITLE	Year 13 Scholarship
TIME ALLOWED	FIVE hours with a break for lunch at the discretion of the supervisor
QUESTIONS	There are TWO questions in the paper. Candidates are to answer BOTH questions. Answer as much of each question as you can. Note that Question 2 is significantly more difficult than Question 1. Plan your time to allow a good attempt at each.
INSTRUCTIONS	Candidates may use any text or manual or online programming language documentation for reference during the examination. Candidates may not copy code from the internet or consult anyone other than the examiners during the examination
DETAILS	Both questions pose problems which you are asked to solve by writing computer programs. They may also ask for written answers for some problem parts. In programming you may work in the programming language of your choice. However, the examiners need to be able to read your program text and if at all possible, test run it. If problems arise from your choice of programming language, we may contact you after the examination, for clarification. Written answers to parts of questions can be submitted in text files; included as comments in your program text; or as photographed or scanned images of hand written documents. Remember also that partial marks may be awarded for programming ideas written down, but not yet implemented.
CALCULATORS PERMITTED	Yes

1. **Heartrate** (Careful and Accurate Programming)

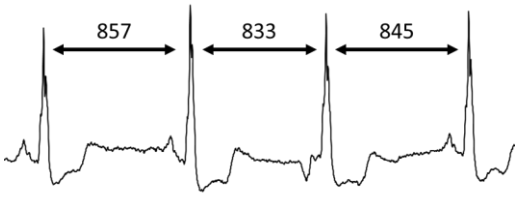
Your programming work in this question will be assessed on three criteria:

(a) Completeness and accuracy of the program. It may be that this problem statement does not state exactly what the program should do under all circumstances. If you find a situation of that nature, choose a solution and write down, either on paper or in the comments of your program what the difficulty was and how you chose to resolve it.

(b) Good presentation. That is, it should make good use of programming language facilities, be well organised, neatly laid out, and lightly commented.

(c) Careful checking. Wherever possible check input from the program user in case they have made errors.

In this question you are asked to write a program that calculates your heartrate. Your heartrate is calculated as 'beats per minute' (bpm). Bpm can be calculated by counting the number of times your heart beats in 1 minute, or by taking the interval between two beats (inter-beat interval) and extrapolating it out to the number of beats per minute. Your program will do the latter.



For example, the picture above shows the electrical activity of the heart. There are four peaks, which represent four heartbeats. The interval between each beat is called the inter-beat interval. We can use the first inter-beat interval to calculate bpm as follows: 60,000/857 = 70 where 60,000 is one minute in milliseconds, 857 is the first inter-beat interval, and 70 is the resulting bpm (rounded to the nearest whole number).

You can use one inter-beat interval at a time to calculate bpm. However, if the data is noisy, this can result in some extreme differences. Instead, you should 'smooth' the bpm by using more than one inter-beat interval at a time. You can do this by using: (1) the cumulative mean, (2) a sliding window, or (3) a dynamic sliding window. The cumulative mean is simply the mean of all inter-beat intervals up to and including the current one, i.e. [1], [1,2], [1,2,3], [1,2,3,4], etc. A sliding window calculates the mean inter-beat interval from *n* number of intervals, moving across by one each time, i.e. [1,2,3,4,5], [2,3,4,5,6], [3,4,5,6,7], etc. A dynamic sliding window starts by calculating the cumulative mean until there are enough elements to begin the sliding window, i.e. [1], [1,2], [1,2,3], [1,2,3,4], [1,2,3,4,5], [2,3,4,5,6], [3,4,5,6,7], etc.

Your task is to write a program which interacts with a user allowing that user to enter a series of inter-beat intervals. It should use these intervals to calculate the user's heartrate, as bpm rounded to the nearest whole number. Your program should calculate the bpm in four different ways, as follows.

- ⇨ Using individual inter-beat intervals (as shown in the example above)
- ⇨ Using the cumulative mean of inter-beat intervals
- ⇨ Using a sliding window of inter-beat intervals (with a window size of 5)
- ⇨ Using a dynamic sliding window of inter-beat intervals (with a maximum size of 5)

QUESTION 2 CONTINUES ON THE NEXT PAGE

Finally, your program should also display the heartbeats visually (see sample output below), both for bpm calculated using individual inter-beat intervals, and for bpm calculated using the dynamic sliding window. For example, for the former, if the bpm is 71, you will display 70 dashes and one plus sign. For the latter, if the bpm is 75, you will display 74 dashes and one asterisk.

- ⇨ Display bpm calculated using individual inter-beat intervals
- ⇨ Display bpm calculated using a dynamic sliding window

The transcript of a sample interaction with such a program is given below. In the transcript, information entered by the user is shown in **bold** type. You don't have to follow this style of data entry or format results in the same way. The sample is just here to show the kind of interaction expected of your program.

```
Heartrate Calculator

How many inter-beat intervals would you like to enter: 10
  Enter interval 1: 857
  Enter interval 2: 845
  Enter interval 3: 833
  Enter interval 4: 857
  Enter interval 5: 833
  Enter interval 6: 822
  Enter interval 7: 800
  Enter interval 8: 811
  Enter interval 9: 789
  Enter interval 10: 769

Calculating Results

  Would you like to calculate your heartrate using individual intervals (y/n): y
    70
    71
    72
    70
    72
    73
    75
    74
    76
    78

  Would you like to calculate your heartrate using the cumulative mean(y/n): y
    70
    71
    71
    71
    71
    71
    72
    72
    73
    73

  Would you like to calculate your heartrate using a sliding window(y/n): y
    -
    -
    -
    -
    71
    72
```

QUESTION 2 CONTINUES ON THE NEXT PAGE

```
  72
  73
  74
  75

Would you like to calculate your heartrate using a dynamic sliding window(y/n): y
  70
  71
  71
  71
  71
  72
  72
  73
  74
  75

Would you like to display your heartrate(y/n): y
  ------------------------------------------------------------------------+
  --------------------------------------------------------------------------+
  ----------------------------------------------------------------------------+
  ------------------------------------------------------------------------+
  ----------------------------------------------------------------------------+
  ------------------------------------------------------------------------------+
  --------------------------------------------------------------------------------+
  ----------------------------------------------------------------------------+
  ------------------------------------------------------------------------------+
  --------------------------------------------------------------------------------+

Would you like to display your smoothed heartrate(y/n): y
  ------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  ----------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
  --------------------------------------------------------------------------*
```

QUESTION 2 CONTINUES ON THE NEXT PAGE

2. **Images** (Problem Solving and Programming)

Your programming work in this question will be assessed on three criteria:

(a) Your approach to the problem. We will be looking at your work for evidence that you found good ways of storing the necessary data, and devised algorithms for finding and displaying the requested results. ***Please hand in any notes and diagrams which describe what you are attempting to program, even if you don't have time to code or complete it. You may include comments in your program, or write a description of your program to hand in.***

(b) The extent to which your program works and correctly solves the problem.

(c) The extent to which you use results from your programming to explore the problem presented.

You may find that the programming language you use makes it difficult to produce output as shown in the example implementation steps below. If this is the case, feel free to build your program in a way that suits your circumstances.

Working with photographs or other images is a common activity on modern computers. In particular, graphics capabilities enable us to view detailed images. Early computers didn't have graphics displays; they were only capable of displaying text. Text displays can be used for (very) low resolution graphics, sometimes using the shapes of text characters as part of the images. Here are two cat examples of 'ASCII art'.

```
|\---/|          _._     _,-'""`-._
| o_o |         (,-.`._,'(       |\`-/|
 \_^_/              `-.-' \ )-`( , o o)
                          `-    \`_`"'-
```

In this question, you are asked to write a program to display, explore and process image data using only a text display. The question presents the problem in stages for you to program. The stages include further explanation of the way in which the large amounts of data in an image can be reduced for low resolution display, and of the operations required. We suggest that you build your program in the order given. This will make it likely that you have parts working at the end, even if you don't have time to complete the whole program. However, we also strongly suggest that you read through all the stages before starting to program. Stage I is the final stage, in which you have the most freedom to explore algorithm ideas.

The stages of this problem involve building and changing a program. Instructions will be given in some detail for the first stages. Later stages require that you develop the code yourself. When you are making a major change, you should save a working version of your program. This will help us see what you have achieved, especially if you have difficulties with the altered version. Where stages ask you to try different ways of displaying an image, you can write different display procedures within the same program to make sure that all of your answers are still visible to the examiner.

QUESTION 2 CONTINUES ON THE NEXT PAGE

Image Data Storage

An image is a grid of pixels, each with a colour. For example, consider the following picture of an apple. It is made of 216 columns each of 217 rows of pixels (46872 in total).

From left to right, the images below magnify an area near the base of the stalk more and more. In the right hand image especially it can be seen that the picture consists of small square (pixels) of colour.

Computer displays generate colours by mixing various amounts of the three primary colours: red; blue; and green. In an image therefore, each colour is stored as three numbers, being the levels of each colour component. The colour levels are each stored as values in the range 0 to 255 (so as to fit into one byte of memory). Some examples:

- A pure bright red pixel might be stored as red=255, blue=0, green=0; meaning full intensity red, with zero for both blue and green.
- A pure bright yellow pixel might be stored as red=255, green=255, blue=0; meaning full intensity for both red and green with no blue (red and green light combine to give yellow).
- Red=0, green=0, blue=0 gives a black pixel.
- Red=255, Green=255, Blue=255 is a white pixel.
- A mid pink (half way between red and white) might be red=255, green=128, and blue=128.

A wide variety of file formats are used for storing images on computers. You have probably used .bmp, .png and .jpg formats at some time. For this question we have devised our own image format. An image is stored in a text file. The first line has two numbers separated by

QUESTION 2 CONTINUES ON THE NEXT PAGE

a space. These numbers are the width and height of the image (in pixels). After that there are lines for each pixel starting with the leftmost pixel on the first row, then the second pixel along the row, and so forth to the end of the first row. The next line has the left most pixel of the second row and so forth to the end of the image. The first few lines of the apple image file are as follows, showing a width of 216 pixels and a height of 217 pixels. The six pixel colours shown are all the same off-white colour from a corner of the image. There are 46,873 lines in the file.

```
216 217
243 239 236
243 239 236
243 239 236
243 239 236
243 239 236
243 239 236
. . .
```

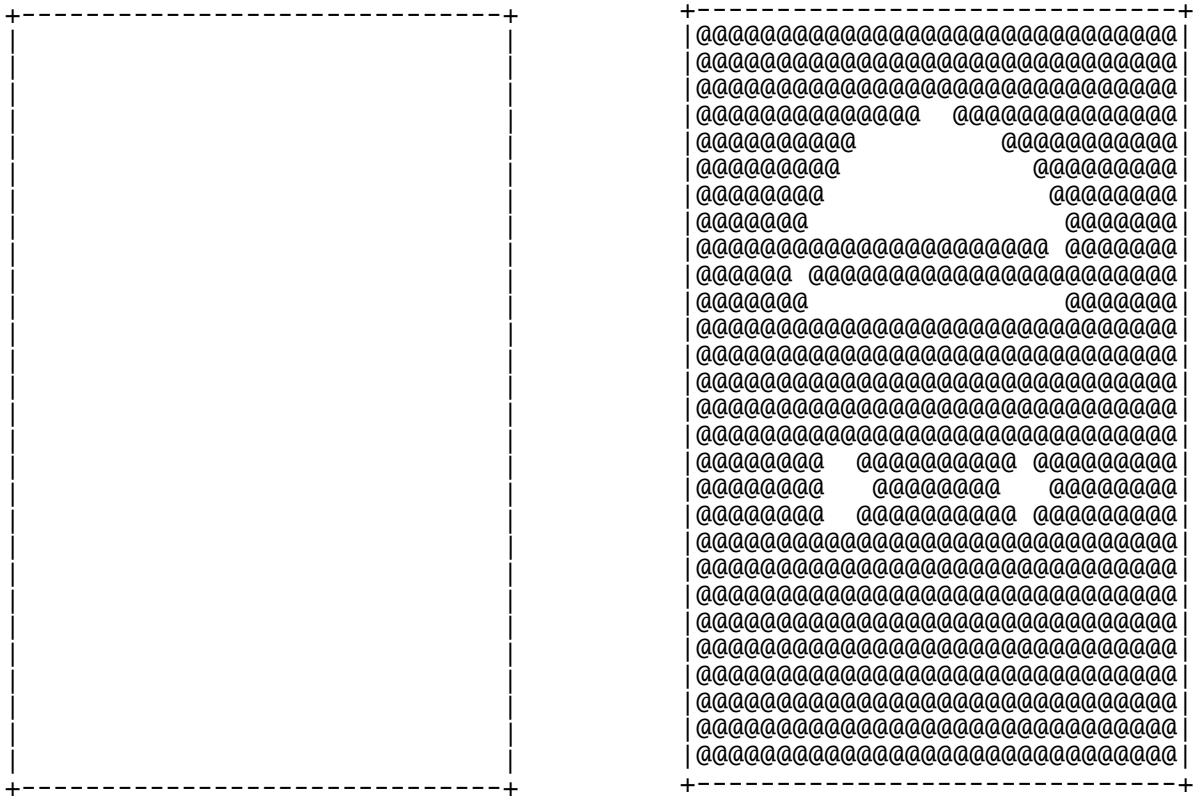
We have supplied a number of sample images for you to use. In each case there is a .txt file which has the data as described above, and a .bmp file so that you can look at the picture using any image viewing software on your computer (usually a double click will open a photo viewer or paint program). Please note that your program should read the .txt file. You are not required to program reading of the .bmp file. The images supplied are

Emoji	A small image with bright colours
Apple	The image shown above
Apple2	Another picture of apples and leaves
Apple2At30pc	A version of Apple2 scaled down to 30% of original size
NZFlag	The New Zealand flag

QUESTION 2 CONTINUES ON THE NEXT PAGE

Stage A: A Picture Frame.

Use the Emoji file for this stage. Look at the file to find out how many rows and columns there are in the image. Write a program to draw a box using text characters. The box should be the right size to hold the Emoji image. The result should be similar to the left part of the figure below.



Stage B: Seeing Something.

Extend your program to read data from Emoji.txt. Our goal is to work towards displaying enough from the image to make it somewhat recognisable. Of course the display resolution is very low and there is no colour, so the best we can hope for is to see shapes. For this first display experiment draw a '@' character when the green intensity is greater than 128 (the middle of the range) and a space (blank) character otherwise. This gives a very rough approximation to a black and white image. (As the human eye is most sensitive to green light, the green level is not a bad approximation to the level of light or dark.) You should get a result like that shown on the right above.

Stage C: The Right Way Up

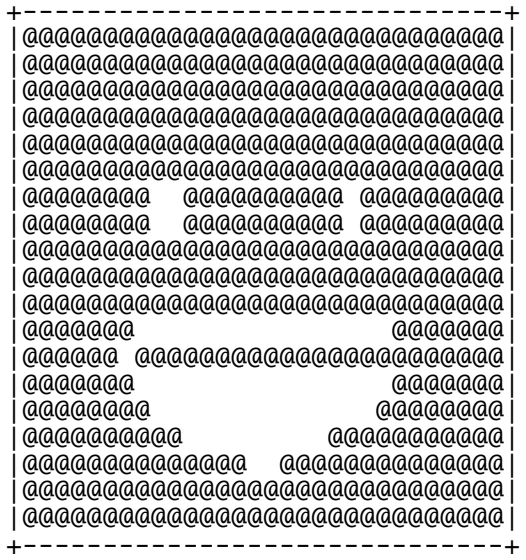
The result of the last stage should be recognisable as having shapes from the Emoji image. It is however upside down (because the rows are stored from bottom to top). Modify your program to display the image the right way up.

Stage D: Scaling

The Emoji image displayed in stages C and D is tall and thin. The original image was close to being square. The problem here is that the image expects square pixels and our 'pixels' (characters) are not square – they are taller than they are wide. We can't easily change our

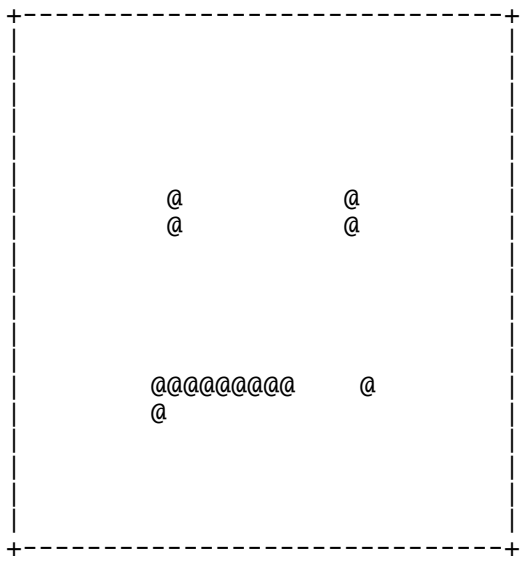
QUESTION 2 CONTINUES ON THE NEXT PAGE

character display. Perhaps a different font and line spacing might work, but this project is going to need a scaling system for dealing with large images any way. Find a way of shrinking the image vertically. *Hint: The following example simply leaves out every third line of the image.*



Stage E: Colours

The Emoji image was drawn using the colours red, yellow, black and white. If we modify the program to just show black parts the result might be as follows:



Your task here is to produce four versions of your program to display just black (as above), just red, just yellow and just white. *Hint: A red pixel has a high red intensity, but so does a white pixel. Red pixels have high red, but low green and blue.*

Stage F: More Scaling

Scaling becomes more important when we deal with larger images. Starting with the Apple image, produce a version of your program which can display the image in close to the correct shape, but at a scale that fits on a screen for easy viewing. Try each of the other images and choose appropriate scales for each.

Stage G: Colours Again

The colours used in the Emoji image were bright and clear. The colours in the Apple image are more subtle. Write a version of your program (using scaling developed in Stage F) which displays the mostly red areas of the apple.

Stage H: Different Colour

Adapt your program from Stage G to display the red areas from the second apple image (you may use either version of the image).

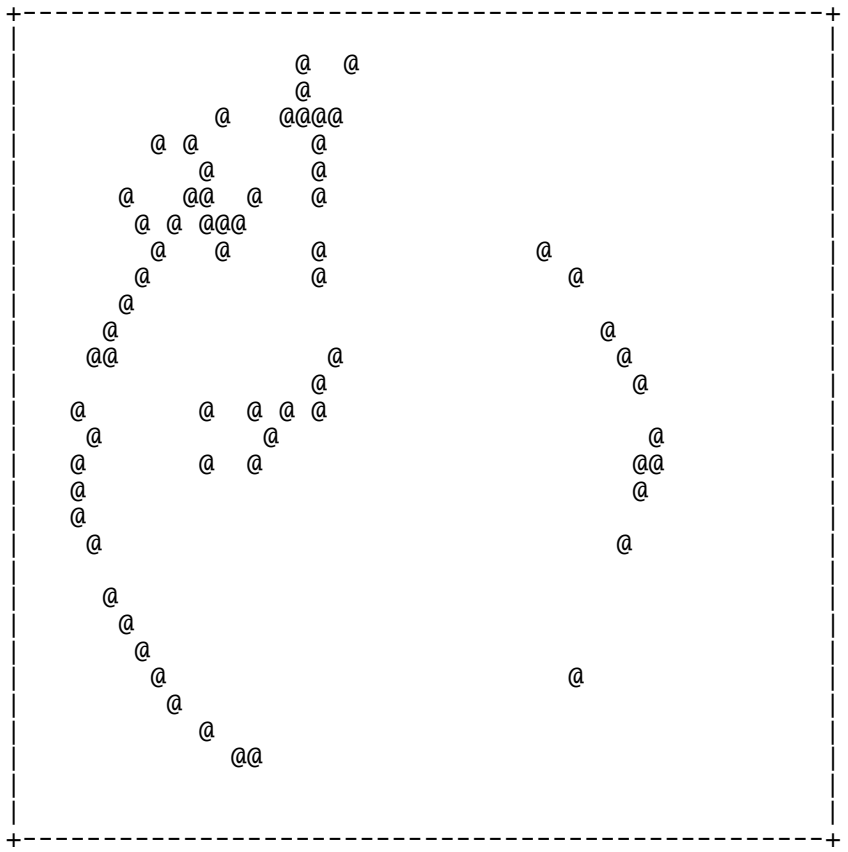
Edge Detection

When we draw with pencil on paper we often draw outlines for objects. For example, the basic shape of the apple and its shadow are captured in the sketch below (done by manually tracing around the shapes in the image). In image processing, 'tracing' around edges (which we call 'edge detection') is often used as a first stage when recognising objects in a scene. The idea is to make a new image in which we introduce bits of line whenever the colour or brightness in an image changes sharply between nearby pixels.

More specifically we generate an image in which we set a pixel to be part of a line if the pixels near it differ in colour. We have choices to make in deciding exactly which pixels and colours of pixels to compare, and finding a solution which works well on our low resolution images is your task. The following is a simple algorithm which finds some 'edges', but not all. It decides what to display at each pixel position as follows:

if the pixel position is on the left or right edge of the image
    set the output to " "
else if the pixel to the left and the pixel to the right of the position being considered have green intensities that differ by more than 32
    set the output to "@"
else
    set the output to " "

The result looks like this:



Note that we see edges corresponding to the sides of the apple. However, some choices were made in writing the algorithm – we are only comparing pixels to the left and the right; we are only looking at the green colour component; and we are looking for differences greater than 32. Two effects are: we mostly see vertical lines because horizontal lines would occur when the pixels above and below a given point differ. Detection of the boundary between the apple and its shadow is poor.

Stage I: Edge Detection

Your task is to improve the edge detector, and try it on the images provided. Your answer should include a written description of the method(s) that you try.