159.102 Instructions for Assignment 3

Assignment 3 starts in Week 9 and is due in Week 12 (20 October 2023).

NOTE: Assignment 3 counts 15% towards your final result.

It is a good idea to put your name and ID number in a comment at the top of your program.

You have been given a contract by a factory that produces buttons. It is important that the factory identifies damaged buttons so that they are not supplied to the stores. The factory has a camera that takes a photo of buttons. The camera works only in black and white (no colour) and the resolution is not very good, but that is not a problem.

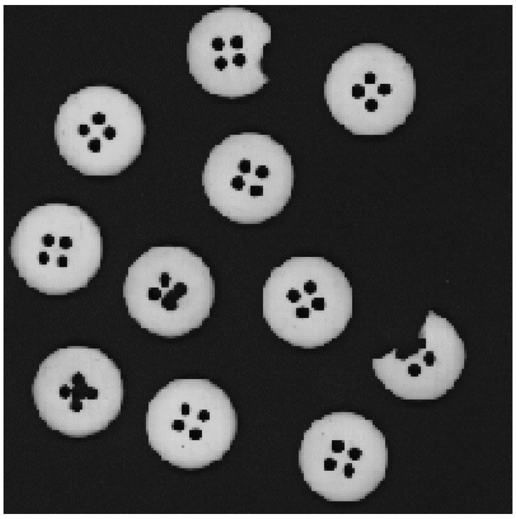
Your job is to write a C++ program that identifies any damaged buttons in the photo. You need to produce an image that displays a box around each button. If the button is damaged you must display a red box and if the button is not damaged you must display a green box. Make sure you read carefully through all the sections below.

Section A - input

The input to your program is the photo taken by the camera in the factory. This is available in a .ppm file called Buttons.ppm. This file is available under Assessments on Stream. Do not edit this file in any way. If you accidently modify the file then download a fresh copy of the file from Stream.

Your program must be able to work with any such photo. Do not assume a specific number of buttons. do not assume that buttons will always be in the same place in the photo.

(*Hint: Before starting on your program, check that the .ppm file has no errors. Download Buttons.ppm from Stream and convert it to .bmp (or other format) and look at it. The display should look like this:*



Just for interest – you can tell that the resolution of the camera is low because of the "stepped" edges to the buttons in the image. Actually, for many problems of this type (i.e. identifying defects in products) it is often better to use a black-and-white photo because the defects stand out more clearly.

Section B – understanding the problem

Like many "real-life" systems, this type of project can never be perfect (which is what makes real-life projects interesting). We do the best we can by noting the following:

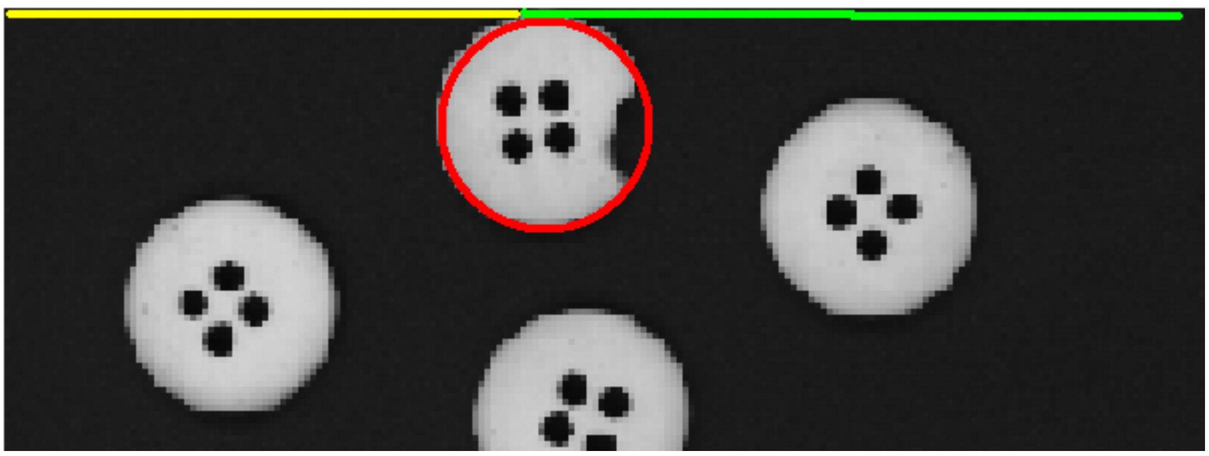
Notes about the problem:

1. Buttons appear as white (or light grey) objects on a dark background. This is a black-and-white photo which means every pixel is a shade of grey (i.e. the R, G and B values are the same for each pixel). We define a pixel to be part of a button if its R (or G or B) value is greater than 128.
2. There will always be a few pixels around the edge of a button (depending on the shadows) that are darker than this and will thus not count as part of the button. This does not matter.
3. We need to know how to "identify" a button. Basically we look for pixels where the R value is greater than 128. But we need more than this – see next section below.
4. To draw a box around a button you need to know the minimum and maximum x-value of all pixels in the button and also the minimum and maximum y-value of all pixels. The box then has a top left corner of (xmin, ymin) and a bottom right corner of (xmax, ymax) and so on.
5. Some thought needs to be given to how we define a "damaged" button. This is entirely up to you. *Hint: a damaged button will have less total pixels than an undamaged button.*

Section C – the algorithm to identify a button in the image

A button consists of pixels with R value greater than 128 AND the pixels must touch each other. If we work through every pixel, we can identify a button by:
a) finding a pixel with R value greater than 128
b) finding all other pixels that connect to that pixel (and have R value greater than 128)
c) go back to where we were in (a) and continue

The image below is trying to show this. Step (a) is shown in yellow. We start at the top left and work steadily across and down the image until we find a suitable pixel. Step (b) is shown in red – we find all pixels connected to the first one. Step (c) is shown in green – we go back to where we were at step (a) and continue looking for pixels with R value greater than 128. Note that this diagram is to get the idea – the drawing is not perfect.



Now let us look at step (b) in more detail – if we have one pixel in the button, how do we find the others?

Assume that we have found pixel A at location (x, y) with an R value of greater than 128. Thus we know that pixel A is inside a button. We can then work through the pixels that touch pixel A (they are pixels B, C, D and E). Note the locations of these pixels. Pixel B is in the same row of the image as pixel A so has the same y value. But pixel B is one place to the left so it has a x value of x – 1. Pixel E is in the same vertical column of the image as pixel A so has the same x value. But pixel E is one place further down the screen so it has a y value of y + 1. Similarly for the other pixels.

	Pixel D **(x, y - 1)**	
Pixel B **(x - 1, y)**	**Pixel A** **(x, y)**	**Pixel C** **(x + 1, y)**
	Pixel E **(x, y + 1)**	

Now that we know how to identify the "next door" pixels of pixel A, we have an algorithm as follows:

```
Find pixel A at location (x, y) and look for all connected pixels by:
        Find pixel B at location (x – 1, y) and look for all connected pixels;
        Find pixel C at location (x + 1, y) and look for all connected pixels;
        Find pixel D at location (x, y – 1) and look for all connected pixels;
        Find pixel E at location (x, y + 1) and look for all connected pixels;
```

This is a **recursive algorithm** – find the pixels connected to A by finding the pixels connected to B, etc.

While this may not look like the famous "caves" program, it is essentially the same situation. And we have the same problem. We could develop an infinite loop where pixel A checks pixel B which checks pixel A which checks pixel B, etc. And we solve the problem in the same way, i.e. we put a boolean into each pixel and as soon as we have checked a pixel we exclude it from the search. Do not check that pixel again.

Every recursive function needs a **base case**. In this case there are two which are:
- return if the pixel you are checking has an R values of 128 or less
- return if this pixel is excluded from the search (i.e. if this pixel has been checked before)

Some astute readers may have noticed that we are going on as if they are FOUR pixels next door to pixel A when in fact there are EIGHT next door pixels. We left out the diagonal pixels. The reason for this is that the recursion eventually works its way through all adjacent pixels. E.g. the pixel that is up and to the left of A is also above B so will be checked when B is checked.

Section D – program design

There was another programmer who used to work at the factory. Unfortunately, that programmer did not study 159.102 at Massey and was therefore unable to complete the project. You may find some interesting ideas in the partially completed program which is called Ass3-start.cpp and is available under Assessments on Stream.

Download the program called Ass3-start.cpp and study it.

You **MUST** use the class called pixel_class. Note that two of the methods are at the end of the program. This class is as discussed in the notes but has an extra boolean variable called exclude to assist with the recursive function. This exclude variable is set to false at the start of the program and is set to true if this particular pixel has been checked.

You **MUST** use the following global variables:

```
int screenx, screeny, maxcolours;
pixel_class picture[600][600];
```

It is highly recommended that you also use the global variables:

```
int total, xmin, xmax, ymin, ymax;  // these MUST be global
```

However, if you make the program work without these variables then you do not need to use them.

You **MUST** use the function called `loadButtons` exactly as it is in the program.

The rest is up to you. You can keep everything currently in the program or replace some of it as long as you use the compulsory sections of code listed above.

The basic outline of the main program is as follows:

- load the photo data into the picture using the function `loadButtons`
- work through all pixels identifying buttons and placing boxes into the picture
- write the picture data to a new .ppm file
- (outside your program) convert your new .ppm file to .bmp and view it

Extra notes on drawing a box:

You draw a box (or in fact anything) by placing pixels of a particular colour into the picture.

A box needs four values called xmin, xmax, ymin, ymax.

The top left corner of the box is (xmin, ymin) and the top right corner of the box is (xmax, ymin).

The bottom left corner of the box is (xmin, ymax) and the bottom right corner of the box is (xmax, ymax).

To draw the top line of the box, use the following loop (or similar):

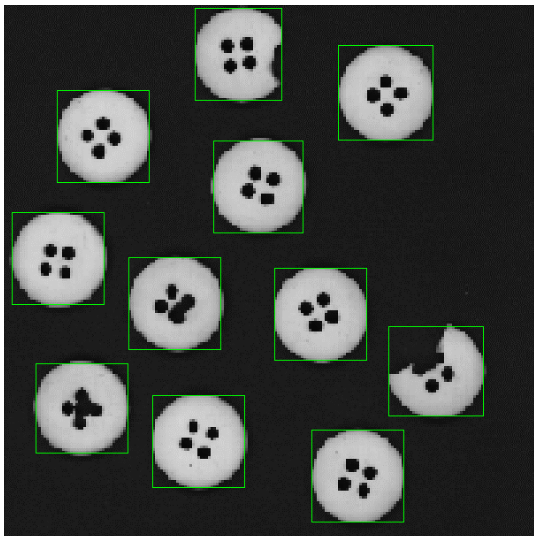
```
for (x = xmin; x <= xmax; x++) {
    picture[x][ymin].loaddata(R, G, B);
    picture[x][ymin].setexclude(true);
}
```

It is **very important** to set the exclude variable in each pixel to true. These pixels are now part of a box and no longer part of the buttons image. They must be excluded from any future searches for buttons.

Section E – output

The output from your program is an image stored in a .ppm file. In order to view the image you will probably need to convert it to a different format, e.g. a .bmp file.

The output image must show the buttons with boxes displayed around each button. The box must be red if the button is damaged and green if the button is acceptable. It should look like this:



Oh dear, this image shows only green boxes. This is not the correct result.

Note 1: if you look very closely, you may see that some boxes do not perfectly sit around the button. There may be one or two pixels on the "wrong side" of the green line. Do not worry about this. Do not waste hours of time trying to get your boxes better than what is shown above. These boxes are perfectly adequate to show which button is being referred to.

Note 2: you need to decide what defines a "damaged" button. Some buttons are obviously damaged, others may be ok, not quite sure about that. Welcome to programming in the real world! As long as the obviously damaged buttons are classified as damaged, that is ok. There may be one or two buttons that some people may regard as damaged and other people may not. In the real factory, the "damaged" buttons are checked by human experts before being discarded.

Some general notes about the assignments in 159.102

- You can find the assignment instructions in a file under Assessments.
- You submit your assignments via Stream (under Assessments) before the due date and time
- The due date and time appear on the Assignment under Assessments (where you submit)
- **Submit only your .cpp file**

- Do not submit the .exe file or any data files or screen shots of the program running
- Staff are not available to check your assignment before you submit it.
- Do not rush into submitting an assignment. You may find useful information in the notes during the week after the assignment starts.
- Assignments may use C++ knowledge from 159.101, 159.102 and elsewhere. However, if you use knowledge from elsewhere, make sure you use it correctly.

IMPORTANT rules for assignments in 159.102

- You may get assistance when writing an assignment. Assistance includes asking questions and getting ideas from teaching staff and other students. Assistance also includes asking for help when your program is not working correctly and you cannot find the error.
- You may **NOT** get someone else to write your assignment for you. If you submit a program written by someone else, you will lose a significant amount of the marks for the assignment.
- You may **NOT** copy a program from the internet. If you submit a program copied from the internet you will receive **ZERO** marks for the assignment. It is very easy for markers to find the same program on the internet.
- The important thing is that you must show that you understand what is happening in the program you submit. Teaching staff will sometimes arrange zoom sessions with students to check that they understand their submission. If this happens to you, please do not be offended – it is something we have to do as part of the quality assurance for the course.

Working on your assignments in 159.102

- You need an editor/compiler to create and run your program. VSCode is provided (see notes to install VSCode under Week 1) but you can use any other IDE that supports C++
- Build up your program, for example: start by only converting decimal to binary. When this is working include binary to decimal. Then build in the error checking.
- Give yourself plenty of time. Do not start 6 hours before the deadline!
- Do not give up just because the deadline arrives. You will still get some marks for a partial solution. In a difficult situation, you can apply for an extension.

Marking criteria for assignments in 159.102

Assignments are marked out of 10 and marks can be lost for:

- programs not compiling or running
- errors in code
- programs that have not been tested for a variety of situations
- programs that do not follow the instructions that are provided
- programs that appear to be written by someone else
- programs that are copied from the internet