Style (12 marks)

Examine the m files and review the following style categories. Select one of the options underneath each heading. The marks awarded for each option are shown in brackets. When reviewing on Aropa remember that if you have marked them down for anything you need to include some written feedback as to why.

Headers for functions (2)

- **Well written headers for all files (2)**
  There is a well written, easy to understand header which describes the purpose of each function file, including the inputs and outputs.
- **Poorly written and/or missing a headers. (1)**
  There are headers but they are poorly written or hard to understand and/or a header is missing
- **Missing two or more headers. (0)**
  There is no description at all for two or more files.

Other comments (2)

- **Well commented (2)**
  The comments clearly describe the purpose of the program and the steps used
- **Commented but not to a great standard (1)**
  The comments use language which is unprofessional and/or are not sufficiently descriptive
- **None (0)**
  There is no commenting in the code (other than the headers)

Indentation (2)

- **Perfect (2)**
  All of the code is indented correctly according to the standard code conventions
- **One file incorrectly indented (1)**
  There is one file which contains lines of code that are not correctly indented
- **Inconsistent (0)**
  There are two or more files which contain lines of code are not correctly indented, according to the standard code conventions

Layout (1)

- **Code is nicely spaced out (1)**
  Code grouped into logical chunks, as seen in Summary Programs.
- **Poor layout (0)**
  No blank lines used to group codes in chunks and/or comment lines are hard to read due to being overly long

Variable names (1)

- **Well chosen (1)**
  All variable names either give a good indication of what the variable is used for, use a sensible abbreviation (with a comment to indicate what is stored in the variable) or follow standard conventions (eg using i for loop variables)

- **Poorly chosen variable names (0)**
  Some variable names are not easily understood and are not well commented

Function names (2)

- **As specified (2)**
  All six required functions are named correctly, using the exact specified names, including the correct spelling and case (GetPatternForValue, GetValueForChar, Code128BChecksum, CreateBarcodePattern, CreateBarcodeImage, ReadPattern)
- **One incorrect filename (1)**
  One of the required functions is not named correctly (the name must match up **exactly**, including case)
- **Two or more incorrect (0)**
  Two or more of the required functions are not named correctly (the name must match up **exactly**, including case)

Code repetition (2)

- **Lines of code are not unnecessarily repeated (2)**
  Loops and/or functions have been used to avoid unnecessary repetition of lines of code. Variables have not been needlessly duplicated.
- **One chunk of code repeated (1)**
  There is one chunk of code which should have been written using a loop, that have been done without one. (eg if four or more lines in a row are identical or similar, you should have used a loop). Alternatively there may be repetition of code from another function (e.g. cutting and pasting code from another function rather than calling the original function)
- **Very similar lines of code repeated (0)**
  There is more than one chunk of code which should have been written using a loop, or made use of an existing function.

Functionality (18 + 3 Bonus marks)

To test functionality, run the supplied master test script. The test script will test each specified function with a range of inputs and compare the outputs against the expected results. This will then generate a mark for each function.

Note that if they have not named their functions correctly you will need to edit the appropriate function name string at the top of the test script. If their function does not take the required input type and return the required output type it will fail the tests and they will get zero marks for that function.

The test script supplied before the due date will be similar but not necessarily identical to the one that will be used to mark the project (in particular the marking script may use a range of images that encode different messages)

A.1 GetPatternForValue (2)

`GetPatternForValue` takes a single input (a code 128B value **number**) and returns a **string** as the single output (the code128B pattern of 1s and 0s associated with the given value).

A.2 GetValueForChar (2)

`GetValueForChar` takes a single input (a **string** containing one character) and returns a single output (the code128B value **number** associated with the string).

A.3 Code128BChecksum (2)

`Code128BChecksum` takes a single input (an array of **numerical** values) and returns a single output **number** (the checksum value, calculated according to the code128B checksum algorithm).

A.4 CreateBarcodePattern (3)

`CreateBarcodePattern` will take a single input (a **string** containing a phrase) and return a single output (a **string** of 1s and 0s representing the barcode pattern).

A.5 CreateBarcodeImage (3)

`CreateBarcodeImage` will take a single input (a **string** of 1s and 0s representing the barcode pattern) and return a single output (a 2D array of **numbers** representing a greyscale image of the barcode that corresponds to the input pattern). Your image should have **60 rows** and the number of columns should match the length of the input string. Your 2D array should contain 1s and 0s as NUMBERS.

B ReadPattern (6)

`ReadPattern` takes a single input (a **string** containing a code128B pattern of 1s and 0s). It returns TWO outputs. The order is important. The first output is a string containing the encoded message. The second output is a Boolean value which is true if the checksum read from the barcode matched the checksum calculated for the message (and is false otherwise).

C ReadPattern (3)

BONUS marks if your ReadPattern function can handle patterns from barcodes which feature the following

- **Barcodes where the narrowest bar is not necessarily 1 pixel wide (1)**
  e.g. it may be 2 pixels wide (or 3 or more).
- **Barcodes where the quiet space either side of the barcode less than 10 times the width of the narrowest bar (1)**
  You can assume there is white space either side of the barcode but it may not be identical in width on each side and it may be less than 10 times the width of the narrowest bar.
- **Barcodes where each bar is not necessarily the identical number of pixels in width (1)**.
  e.g. one narrow bar may be 11 pixels wide whereas the next narrow bar may be 10 pixels wide.