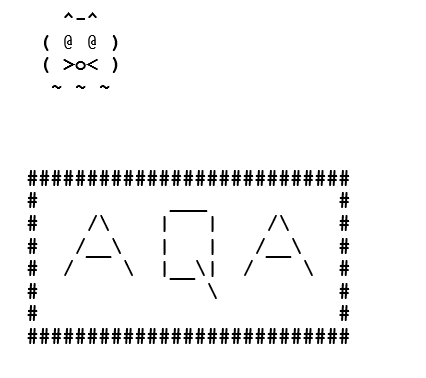
# Graphics – ASCII Art Programmed Solution

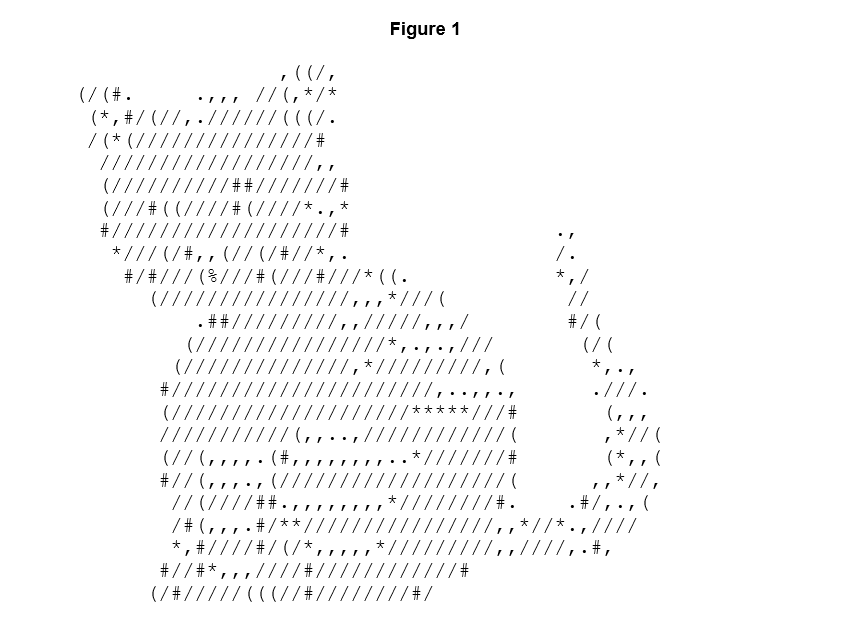
* The programmed solution accompanying this pack is a simple graphics program that uses ASCII art.
* ASCII art is a technique used to represent images using the 95 printable characters in the ASCII character set.
* ASCII art is used where text can be printed but bitmapped graphics cannot, for example in console mode programming environments.
* It requires monospaced fonts (such as Courier New) which position the characters neatly in columns.
* Different techniques can be used in ASCII art for different effects.

Here are some examples:



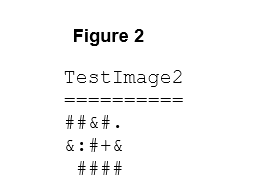
Very elaborate images have been made with different characters emulating different shades for the pixels.

Figure 1 shows an ASCII art image loaded from the image1.txt data file.



The programmed solution uses printable ASCII characters to display images. The program allows the user to load an image from a text file. The option "S - Save image" saves the image in such a way that it can be viewed in a text editor, such as Notepad or TextEdit.

The programmed solution can also load greyscale images where the shade of each pixel is encoded by an 8-bit integer. 8-bit integer encoding allows 256 possible shades of grey. The program changes the greyscale values into ASCII characters. Figure 2 shows the image created when loading the greyscale.txt data file.



The data files ascii.txt, image2.txt and image3.txt available with this Preliminary Material also contain parts of images.

# Tasks

1. **Download the program code and accompanying data files. Ensure all the files are saved in a single directory.**Run the code to experiment with how it works.   
   Open the data files ascii.txt, image 2.txt and image 3.txt. Screenshot them into the table below:

|  |  |
| --- | --- |
| Ascii.txt |  |
| Image 2.txt |  |
| Image 3.txt |  |

1. **Using the table below – describe each procedure or function within the code.**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **Parameters**  *names & data types* | **Return Value**  *data type or n/a* | **Description**  *What does this part of the code do?*  *How does it work?* |
| FileHeader |  |  |  |
| DisplayError |  |  |  |
| PrintHeading |  |  |  |
| DisplayImage |  |  |  |
| SaveImage |  |  |  |
| EditImage |  |  |  |
| ConvertChar |  |  |  |
| LoadGreyScaleImage |  |  |  |
| LoadAsciiImage |  |  |  |
| LoadFile |  |  |  |
| SaveFile |  |  |  |
| ClearGrid |  |  |  |
| DisplayMenu |  |  |  |
| GetMenuOption |  |  |  |
| Graphics |  |  |  |

1. **Programming task: write a new function which will validate whether the chosen symbol to replace actually exists in the current image.**
   1. Write a new function “SymbolExists”.  
      This should search though each character stored in the image grid to see if the character the user has entered is actually in the grid.   
      The function should return TRUE if the symbol is used anywhere within the image grid, and FALSE if it is not.

*Hint: if you need a hand with this task then have a look at the nested for loop used to cycle through the rows and columns to make the replacement.*

|  |
| --- |
| **CODE:** |

* 1. Call the new function “SymbolExists” from an appropriate place inside EditImage. The code to replace a symbol should only execute if the symbol the user is trying to replace can be found in the image grid.

|  |
| --- |
| **CODE:** |

* 1. Test the new function – show a before and after of the image as well as the input/output where you have chosen which symbol should be replaced. You can use any of the test images for this.

|  |
| --- |
| **OUTPUT:** |