Swinburne University of Technology Sarawak

COS10009 Introduction to Programming – Semester 1 / 2018

Function (Lab 06)

Core Task 1

To Do

A) Parameter and Return Value

To explore this topic, we will create a program with a function similar to pow() function in math.h library.

In the main() function, you should:
- ■ Print result for 2 to the power of 1 by calling function mypower(2, 1)
- ■ Print result for 2 to the power of 2 by calling function mypower(2, 2)
- ■ Print result for 2 to the power of 3 by calling function mypower(2, 3)
- ■ Print result for 2 to the power of 4 by calling function mypower(2, 4)
- ■ Print result for 2 to the power of 5 by calling function mypower(2, 5)
- ■ ask the user to enter his/her own base value (int)
- ■ ask the user to enter his/her own power value (int)
- ■ call function mypower() and pass both arguments, base value and power value to the function.
- ■ Display the result of raising base value to the power value input by user.

Function name: mypower

Parameters:
- Base value (int)
- Power value (int)

Other local variables:
- Result of the calculation (int)

Steps:
- Assign the base value into result variable
- Get a loop to repeat according to the power value
- The latest result value will be multiplied by two

Return value:
- Result variable that stores the final result of raising base value to the power value

Core Task 2

To Do

Passing of array as argument to a function

Write a program that will input and store all the individual score of a team of basketball players. The program will then calculate and display the total score of the whole team, it will also display the highest score recorded among all the players. (There are 12 players in a team)

1. In the main function, declare an array with 12 elements
2. Use a for loop to prompt user to input score of each player, and store it into an array
3. Call a function to calculate the total score, then return the total to the main function
4. Call another function to find the highest value in the array. Next, return that value back to the main function
5. Display total score and highest score recorded (the display statements must be done in the main function)

Core Task 3

To Do

SwinGame

The goal of this exercise is to learn a little about how to create a program using the SwinGame Software Development Kit (SDK). SwinGame is a development environment that makes it easy to create programs that use graphics, sounds, animations, networking, and other aspects relevant to creating small interactive games.

When building a program using SwinGame you need to start with a SwinGame Project Template. This template contains the necessary development environment and code that you can make use of.

All the resources of SwinGame is available at **http://www.swingame.com/**

To get started:

1. Download the C_SwinGame_3_6_GCC .zip from Blackboard. This contains everything you need to create your own programs that use the SwinGame SDK.
2. Extract the zip file to your code directory (e.g. C:\users\khsim)

3. Open the ProjectTemplate folder and you should see the following files:

   ■ The clean.sh, build.sh and run.sh scripts. These contain bash commands you will use to clean up the project, compile (or build) the project, and run your program.

   ■ The src folder contains your program's source code

   ■ The lib folder contains the SwinGame code

   ■ The Resources folder contains images, sounds, and other resources you want to use.

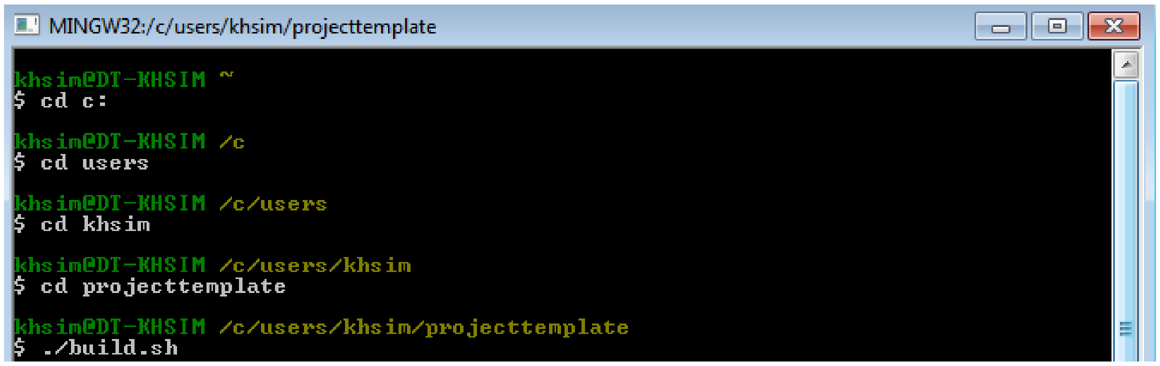
4. Open your windows explorer, browse the following directory:

   C:\MinGW\msys\1.0\msys.bat

   (you may install MinGW compiler at http://www.mingw.org/ if you filed to find MinGW folder in your computer)

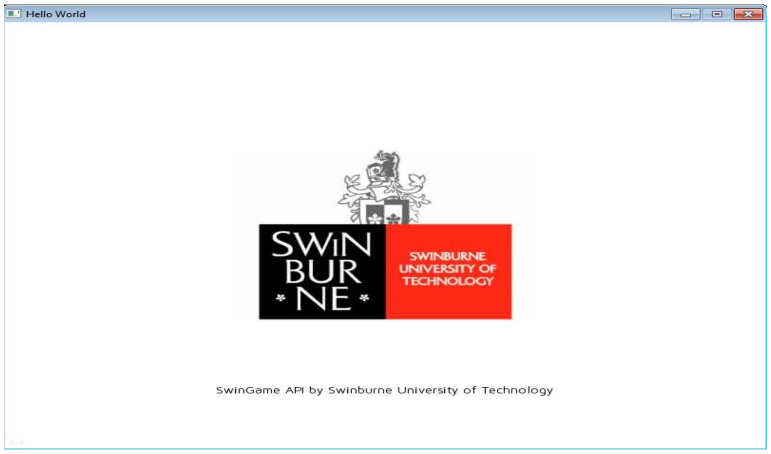
5. Double click to run msys.bat, you should see a console (terminal)

6. Follow the commands shown below to access ProjectTemplate folder, and to compile the start-up project by running build.sh file.



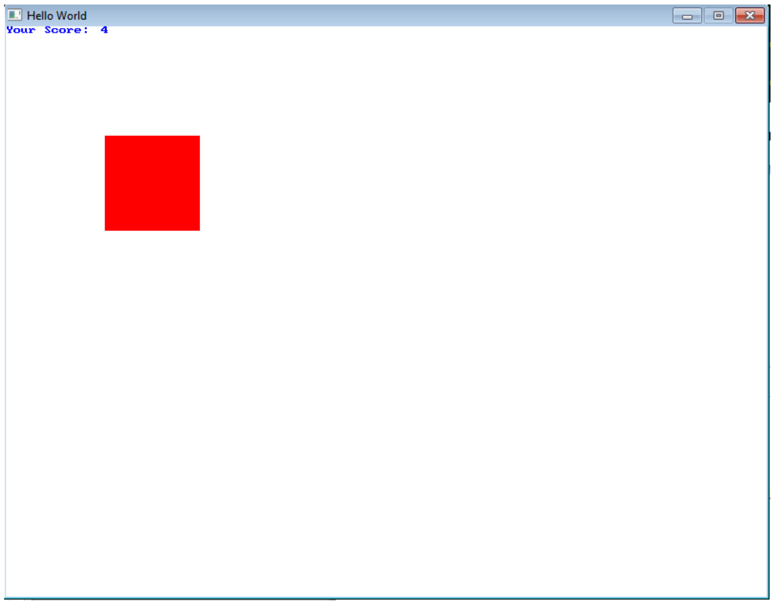
```
khsim@DT-KHSIM ~
$ cd c:

khsim@DT-KHSIM /c
$ cd users

khsim@DT-KHSIM /c/users
$ cd khsim

khsim@DT-KHSIM /c/users/khsim
$ cd projecttemplate

khsim@DT-KHSIM /c/users/khsim/projecttemplate
$ ./build.sh
```

7. Then, type ./run.sh, to run the program. You should see the following as the output:


Hello World
SWIN BUR NE
SWINBURNE UNIVERSITY OF TECHNOLOGY
SwinGame API by Swinburne University of Technology

To complete this task:

1. You are required to edit the main.c file in ProjectTemplate folder to produce a simple game shown below:



```
Your Score: 4
```

2. A red rectangle will appear randomly within the frame for a very short period of time, the user would need to click on the red rectangle by using the mouse cursor. If cursor is clicked within the area of the red rectangle, one point will be counted. However, if the cursor is clicked outside the area of the red rectangle, no point will be awarded.
3. You are suggested to use the following function:

- void draw_rectangle(color clr, float xPos, float yPos, int32_t width, int32_t height);
  http://www.swingame.com/index.php/api/graphics.html#parent_DrawTriangle

- void draw_simple_text(const char *theText, color textColor, float x, float y);
  http://www.swingame.com/index.php/api/text.html

- bool mouse_clicked(mouse_button button);
  http://www.swingame.com/index.php/api/input.html#parent_MouseClicked

Vital Task

To Do

Code reusability / Eliminate repeated code

Lab6Example1.c contains an example program showing un-normalised code. Extract the repeated code, define a function for the repeated code, set up any necessary parameters and call the function as necessary.

Challenge Task

To Do

Mandelbrot

The Mandelbrot set is a collection of complex numbers that can be visualised graphically. This set is infinitely complex, giving complex and visually appealing images when displayed using software.

The values within the Mandelbrot set can be shown graphically as seen in Figure 1. While interesting, the nature of the Mandelbrot set is better explored when color is added to the numbers that lie outside the set. Figure 2 shows a part of the Mandelbrot set, illustrating the complexity of the set.

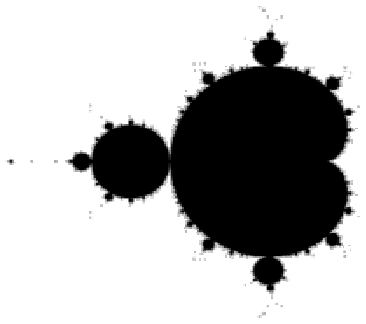


Figure 1: Values in the Mandelbrot set www.ddewey.net/mandelbrot

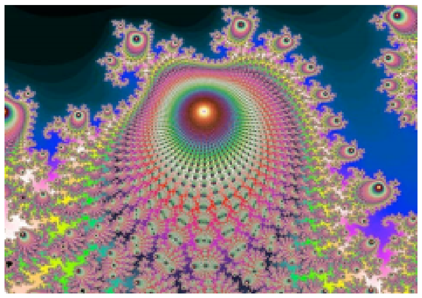


Figure 2: A zoomed in portion of the Mandelbrot set from http://www.linesandcolors.com.

The algorithm to determine if a value is within the Mandelbrot set performs a check to see if $x^2 + y^2$ is less than 2^2, x and y are then projected forward and checked again. This process is repeated over and over, and the value lies within the set when this process can be performed infinitely. To create the visualisations shown, all pixels that lie in the Mandelbrot set are drawn black, with those that lie outside the set are coloured based on the number of times the operation could be performed[1] before the value exceeded the 2^2 limit.

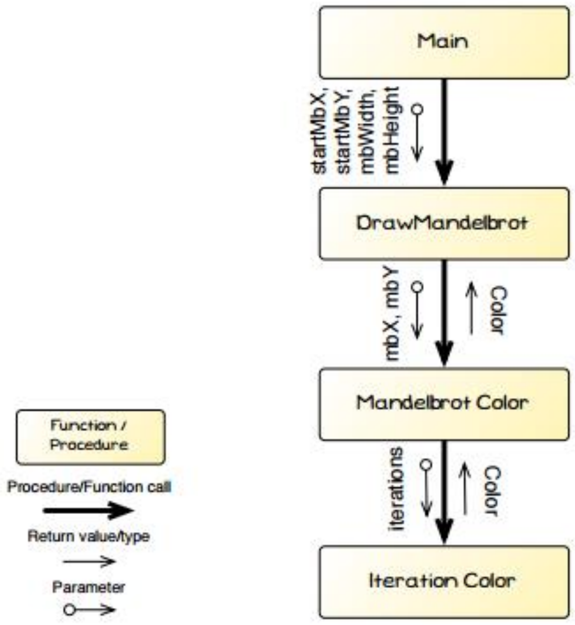
The algorithm used to implement this visualisation of the Mandelbrot set is relatively simple, given the complexity of the output. Read the programmers take on Manbelbrot on wikipedia at http://en.wikipedia.org/wiki/Mandelbrot_set#For_programmers and then see if you can come up with your own program structure to code this. Alternatively, the following pages describe a suitable program structure you can use to implement your own Mandelbrot viewer using SwinGame.

Once you have the code working and showing the full Mandelbrot set, implement zooming when the user clicks the mouse button, allowing them to zoom in on the point selected, or zoom out when they click the right button.

```
Program: Mandelbrot
--------------------
Uses: SwinGame, sgTypes

Const: MAX_ITERATION with a value of 1000

Function: IterationColor
Function: MandelbrotColor
Procedure: DrawMandelbrot
Procedure: Main
--------------------
Steps:
1: Call Main
```

Listing 1: Pseudocode for the Mandelbrot program

The Mandelbrot program has its functionality distributed across four functions/procedures, as shown in the following structure chart. More details of each function and procedure follows.



- The **Main** procedure opens a graphical window and then loop repeatedly until WindowCloseReqested() is true. Each loop this will call ProcessEvents(), DrawMandelbrot(...) and RefreshScreen().
- **DrawMandelbrot** uses the MandelbrotColor function with SwinGame's DrawPixel function to draw the MandelbrotSet to the screen. This procedure will accept four parameters (*startMbX*, *startMbY*, *mbWidth* and *mbHeight*). These values represent the area of the Mandelbrot set that will be drawn to the screen. To view the entire Mandelbrot set you need to pass in values *startMbX* -2.5, *startMbY* -1.5, *mbWidth* 4 and *mbHeight* 3, shown in Figure 4.

- A **MandelbrotColor** function that accepts two parameters (*mbX* and *mbY*) and returns a color. The *mbX* and *mbY* parameters represent the coordinates within the Mandelbrot set space, and will be floating point values (Double). These values will determine if the *mbX,mbY* point is within the Mandelbrot, and based on this determine the color at the indicated point.
- An **IterationColor** function that accepts the number of iterations (integer) and returns a color. This function will be used in the *MandelbrotColor* function to calculate the color of a given iteration value.

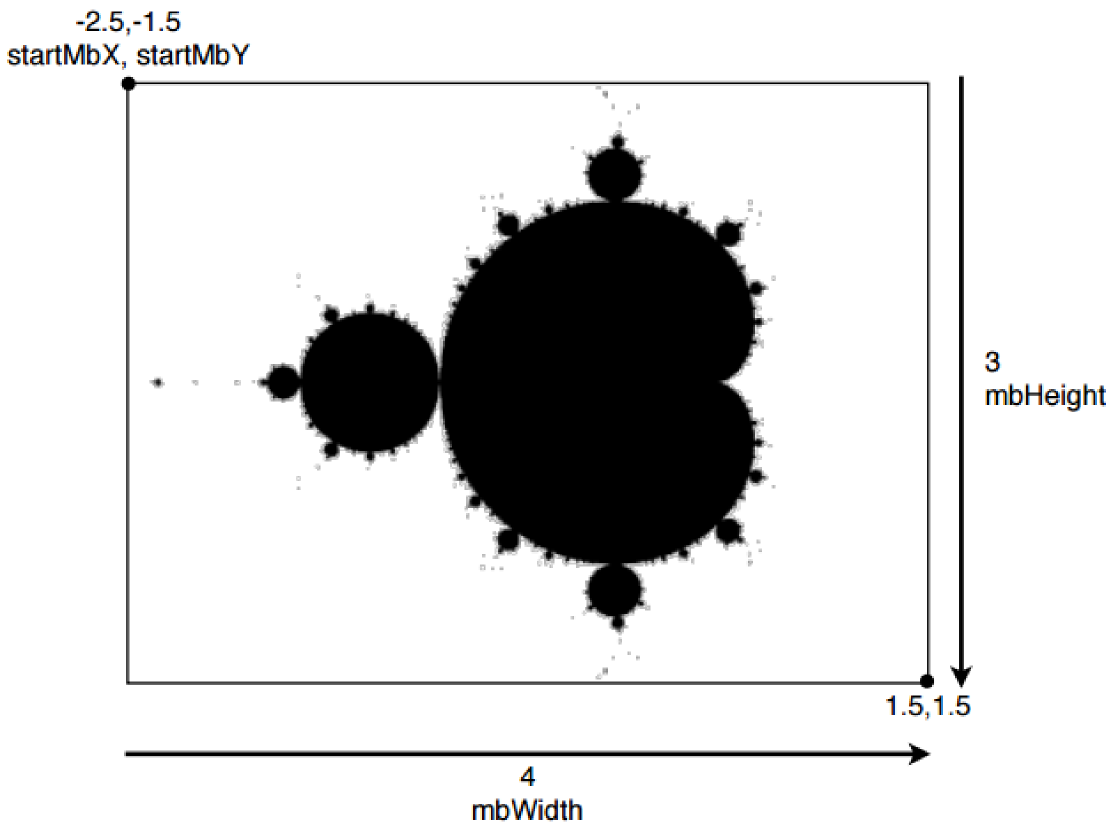


Figure 4: The space occupied by the Mandelbrot set.

```
Function: Iteration Color
--------------------
Returns: a Color
Parameters:
 - iteration (Integer), the number of iterations performed
--------------------
Local Variables:
 - hue (Double) to store the hue of the calculated color
Steps:
 1: If iteration >= MAX_ITERATION Then
 2:   Assign result, the value ColorBlack
 3: Else
 4:   Assign to hue 0.5 + (iteration / MAX_ITERATION)
 5:   If hue > 1 Then
 6:     Assign to hue, the value hue - 1
 7:   Assign result, the value of HSBColor(hue, 0.8, 0.9)
```

Listing 2: Pseudocode for the Iteration Color function

Note: Watch the indentation in the above code... the last four statements are in the else branch.

```
Function: Mandelbrot Color
--------------------
Returns: a Color
Parameters:
 - mbX (Double) the x value in Mandelbrot space
 - mbY (Double) the y value in Mandelbrot space
--------------------
Local Variables:
 - xtemp, x, y (Double) store the altered x,y values
 - iteration (Integer) the number of iterations performed
Steps:
  1: Assign x, the value mbX
  2: Assign y, the value mbY
  3: Assign iteration, the value 0
  4: While ($x^2 + y^2$ <= 4) AND (iteration < MAX_ITERATION)
  5:   Assign xtemp the value $x^2 - y^2 + mbX$
  6:   Assign y, the value 2 * x * y + mbY
  7:   Assign x, the value of xtemp
  8:   Increment iteration by 1
  9: Assign result the value of IterationColor(iteration)
```

```
Procedure: Draw Mandelbrot
--------------------
Parameters:
 - startMbX, startMbY (Double) the location in mandelbrot
space of the top left corner of the screen.
 - mbWidth, mbHeight (Double) the width and height of the
Mandelbrot space to be shown on the screen.
--------------------
Local Variables:
 - scaleWidth (Double) scale of screen to mandelbrot space
 - scaleHeight(Double) scale of screen to mandelbrot space
 - x, y (Integer) screen coordinates
 - mx, my (Double) mandelbrot coordinates
 - mbColor (Color) temporary storage for calculated color
--------------------
Steps:
 1: Assign scaleWidth, the value mbWidth / ScreenWidth()
 2: Assign scaleHeight, mbHeight / ScreenHeight()
 3: Assign x the value 0

 4: While x is less than ScreenWidth()
 5:   Assign y, 0

 6:   While y is less than ScreenHeight()
 7:     Assign mx, startMbX + x * scaleWidth
 8:     Assign my, startMbY + y * scaleHeight
 9:     Assign mbColor, the value of calling
             MandelBrotColor(mx, my)
10:     Call DrawPixel ( mbColor, x, y )
11:     Increment y by 1 // end while y

12:   Increment x by 1
```

Listing 4: Pseudocode for the Draw Mandelbrot procedure

```
Procedure: Main
--------------------
Local Variables:
 - startMbX  startMbY (Double) the location in mandelbrot
space of the top left corner of the screen.
 - mbWidth, mbHeight (Double) width and height of the Man-
delbrot space shown on the screen.
--------------------
Steps:
 1: Open Graphics Window ('Mandelbrot', 320, 240)
 2: Load Default Colors ()
 3: Repeat
 4:   ProcessEvents ( )
 5:   DrawMandelbrot( startMbX, startMbY,
                      mbWidth,  mbHeight )
 6:   RefreshScreen()
 7: Until WindowCloseRequested ()
```

Listing 5: Pseudocode for the Main procedure

1. Copy a new SwinGame project template to your *Documents/Code* directory and rename the folder **Mandelbrot**
2. Implement each of the functions and procedures described above.
3. Switch to the Terminal and change into the project's directory. Compile using **./build.sh** and run using **./run.sh.**

Once you have it showing an overview of the set you need to implement zooming. This should make this much more interesting!

You can implement zoom by altering the values in the *startMbX, startMbY, mbWidth* and *mb-Height* variables. By reducing the width/height you zoom in, by increasing it you zoom out. You need to adjust the startMbX and startMbY to move around in the Mandelbrot set. So zooming involves both changing the size and location you are viewing.

Use SwinGame functions to determine when the user clicked the mouse button (**MouseClicked**) and the position of the mouse at the time (**MouseX** and **MouseY**). With this information you can then alter the *startMbX, startMbY, mbWidth* and *mbHeight* values to zoom in on the area clicked. The illustrations in Figures 5 and 6 are provided to assist you working out how to calculate these new values.

4. Implement zoom by changing the values of startMbX, startMbY, mbWidth, and mbHeight so that the Mandelbrot zooms in when the user clicks with the left mouse button.
5. Switch to the Terminal and compile and run the program - you should be able to zoom in.
6. Add code to zoom back out when the user clicks the right mouse button.
7. Switch to the Terminal and compile and run the program - you should be able to zoom in and out.

Hint: When zooming *in*, make the new mbWidth and mbHeight *half* their current value, and *double* these values when zooming *out*. Other values can be calculated using proportions, see the figures on the following page.

For example, when zooming in:

```
newMbWidth := mbWidth / 2;
```

The location the user clicked would be (using the old mbWidth):

```
startMbX + MouseX() / ScreenWidth() * mbWidth
```

So the new startMbX would be that position, *minus* half of the new mandelbrot width (eg `- newMbWidth / 2`).

The same calculation can be used to determine the y position, and similar steps can be used to zoom out.

Remember to assign the mbWidth the newMbWidth after calculating the new start position.

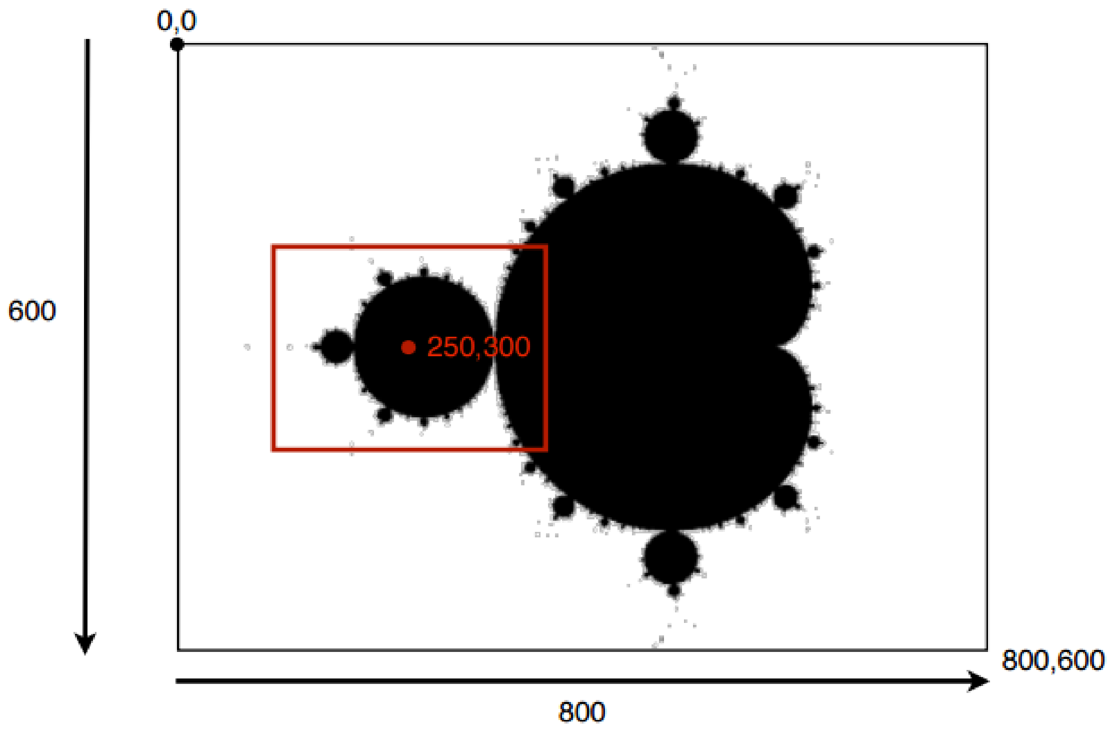


Figure 5: Illustration of the user clicking at 250,300 and the resulting area that will be shown. All values are in pixels (screen coordinates).

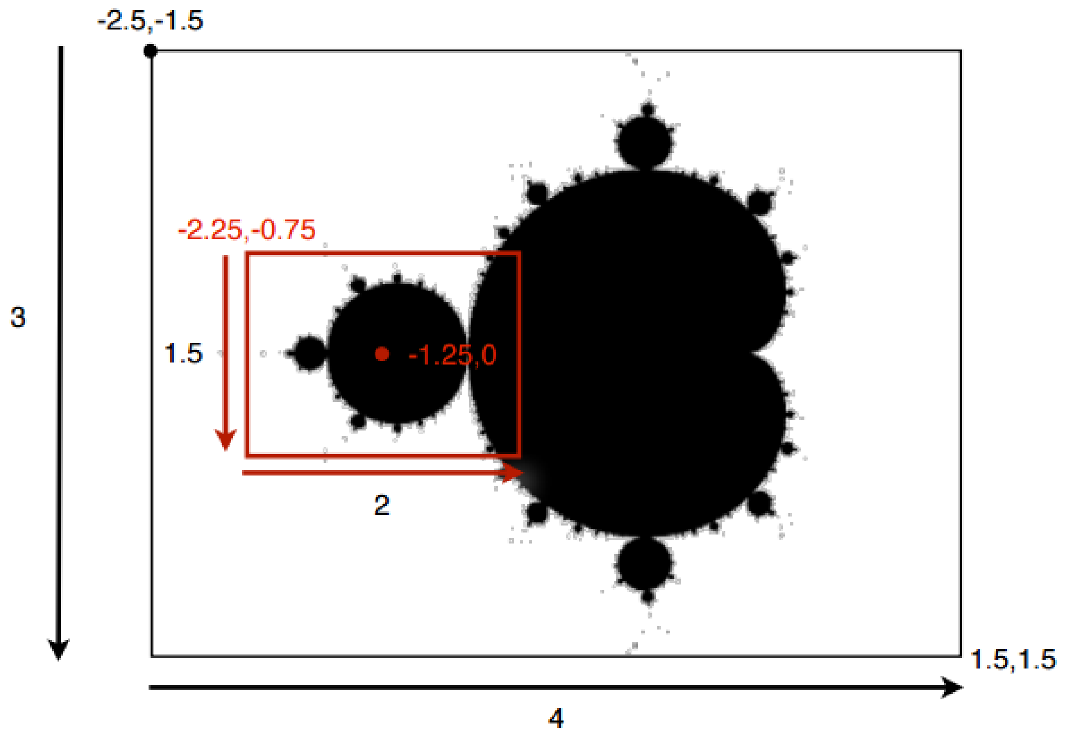


Figure 6: Illustration matching Figure 1 but showing the values in Mandelbrot set coordinates.

<u>Exploratory Task</u>

To Do

Design and write a C program that relates to the topics covered in this lab. The program should demonstrate your ability to use what you have learnt to explore associated concepts that go beyond the scope of this unit.