PRIFYSGOL
BANGOR
UNIVERSITY

Imperative Programming in C

Laboratory 8: Allegro and 2D Graphics

Dr. Llyr ap Cenydd

Introduction to Allegro

Note: This is an optional lab. It will not feature in the module again and is not required for the exam. It is designed to give you some experience building from libraries and coding using graphics. If you are having problems getting Allegro to run, we suggest you concentrate on your assignment for this week.

Allegro is a cross platform, open source, video game and multimedia programming library for C and C++ development. It handles common, low-level tasks such as creating windows, accepting user input, loading data, drawing images, playing sounds, etc. and generally abstracting away the underlying platform.

In this lab we are going to be using Allegro 5. The main aim of the lab is to give experience of importing and using complex external libraries.

Setting up Allegro

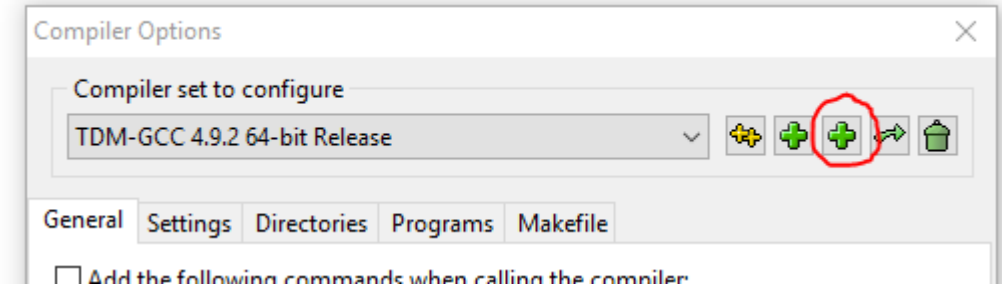
The first thing we need to do is set up our IDE so that it works with Allegro. The idea is we want to reach a stage where we have a "hello world" program that runs successfully, showing an empty window. Once we have reached this stage we will be ready to code!

Note: This technique has been tested on Dev-C++ 5.11. For other platforms we can't offer any support, but there are many tutorials on how to install Allegro on other IDEs such as Visual Studio, Code::Blocks and Linux.

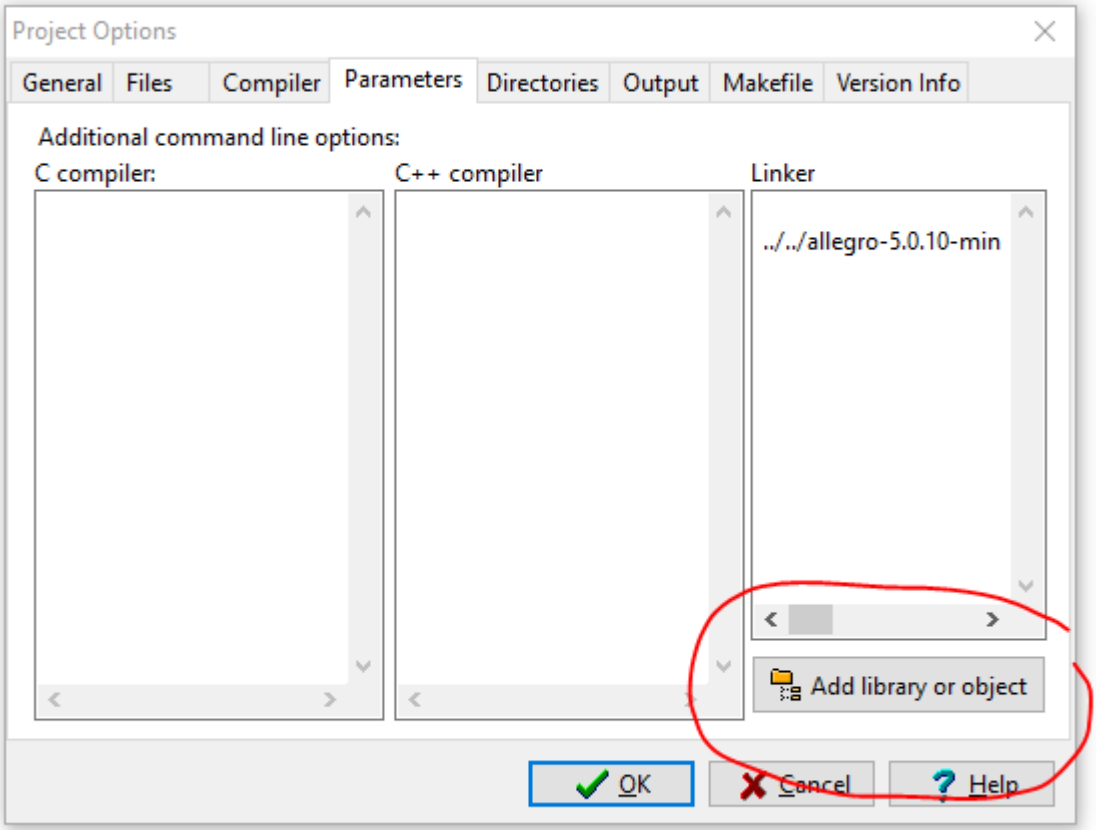
In order to work with Allegro we will need to download the Allegro library, and a compatible compiler.

Allegro 5 (MinGW 4.7.0) (download)
MingGW 4.8.1 compiler (download)

1. Download both files and extract into appropriates folder on your computer. On lab machines you will need to install 7zip from the Windows Store to extract. For lab machines we recommend folders like M:\Temp\Allegro and M:\Temp\MingGW. You might want to delete these folders after finishing the lab to save space.
2. Create a new project in Dev C++.
3. Go to Tools -> Compiler Options. Click on the "Add compiler by folder button".



4. Select the MinGW folder you created in step 1. Click Ok. You should now be able to select the MinGw 4.8.1 compiler.
5. The Allegro folder you created in 1. should contain three folders – bin, lib and include. Go to Project -> Project Options. Go to the Directories tab and “Add” those folders. For “Library Directories” add the path to the lib folder, “Include Directories” add the path to the include folder, and “Resource Directories” add the path to the bin folder.
6. Still under Project -> Project Options, click on the “Parameters” tab and click on the “Add library or object button”.



7. Navigate to the Allegro/lib folder and click on the file called liballegro-5.0.10-monolith-mt.a

8. Copy the code found in Appendix A of this document into your project. You should now be able to compile. If you get compiler problems, double check that the three file paths you added in step 4 are there.
9. To run your compiled exe, you will need to have a dll file in the same folder. Go to the Allegro/bin folder and copy allegro-5.0.10-monolith-mt.dll to the folder where your code/exe is found. The dll (dynamic link library) contains all the Allegro libraries your program will use.

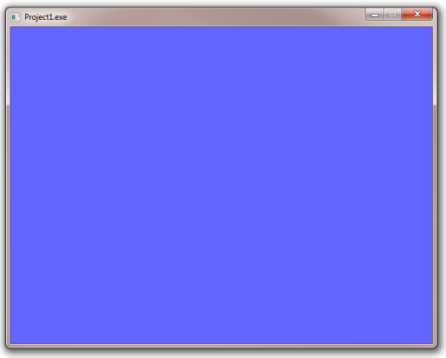
You should now be able to run your basic Allegro program. Currently we are asking Allegro to display a black image on the screen. Find the line:

```
al_clear_to_color(al_map_rgb(0,0,0));
```

and change it to:

```
al_clear_to_color(al_map_rgb(100,100,255));
```

Try to compile->run the code. If everything went ok you should see the following screen:



Congratulations! You have created and edited your first Allegro powered program.

Basic Allegro code

We are not going to look too much on the inner workings of Allegro for this module. We are simply going to learn just enough about how it works so we can draw stuff to the screen. However it's important to know what exactly is happening in the basic allegro shell code that was automatically generated for us. The comments added below should explain what each line is doing. For more information on this and for tutorials on Allegro in general, take a look here.

```c
#include <stdio.h>
#include <allegro5/allegro.h>   //import the main allegro header file

int main(int argc, char **argv)
{
   ALLEGRO_DISPLAY *display = NULL;  //the allegro display variable - allows us to
render to a window

   //initialise allegro, quit if error
   if(!al_init()) {
      fprintf(stderr, "failed to initialize allegro!\n");
      return -1;
   }

   //create the display window, set the resolution to 640 by 480
   display = al_create_display(640, 480);

   //quit if failed to create display window
   if(!display) {
      fprintf(stderr, "failed to create display!\n");
      return -1;
   }

   //Set the back buffer to a light blue color (Red, Green, Blue)
   al_clear_to_color(al_map_rgb(100,100,255));

   //Flip buffer to show on screen
   al_flip_display();

   //rest for 5 seconds
   al_rest(5.0);

   //destroy display
   al_destroy_display(display);

   return 0;
}
```

In the Allegro folder on Blackboard you will find a file called allegro_template.c, which is an extended version of the default code created above and features added functionality such as listening for keyboard and window commands, and an implementation of a game loop.

On Double Buffering

Note: Taken from here.

If you have any experience working with other 2D programming environments, you may have heard of a term called “double buffering”. If you haven’t, here is a little background. Computers are not capable of drawing anything more complex than triangles. Luckily for us, all objects can be represented with a fair amount of accuracy using just triangles. For instance, a square is just two triangles set next to each other. The idea is that the screen is capable of drawing triangles so quickly, that the eye is tricked into seeing whole images at

once. The problem is that a flickering can often be seen because these objects are being erased and redrawn right before your eyes, one at a time. The solution, called double buffering, is drawing all of our images to a hidden, virtual, "screen" which is otherwise known as the **back buffer**. Then, once all of the objects are drawn, the back buffer and the front buffer (the screen) are swapped (using high speed memory transfers). The effect is that the eye sees all of the objects at the same time and there is no flicker. In Allegro 5.0, all of the heavy lifting of creating, maintaining, and utilizing a double buffer system is handled for you. Recall the following code:

```
al_clear_to_color(al_map_rgb(100,100,255));
al_flip_display();
```

What this is doing is coloring our back buffer a nice sky blue, and then swapping the buffers so that the back buffer is now presented onto the screen.

Graphical Primitives

Allegro is capable of drawing any graphical primitive to screen using a single function call. To set up this feature, we first need to import the primitive library:

```
#include <allegro5/allegro_primitives.h>  // primitive header file
```

And initialise the use of primitives like this:

```
al_init_primitives_addon();
```

We only need to do this once, just after creating the display.

Once we've done this, we can ask Allegro to draw a primitive to screen. For example, to draw a circle, square and triangle outline we use the following function calls:

Triangle

```
void al_draw_triangle(float x1, float y1, float x2, float y2,
   float x3, float y3, ALLEGRO_COLOR color, float thickness)
```

Circle

```
void al_draw_circle(float cx, float cy, float r, ALLEGRO_COLOR
color, float thickness)
```

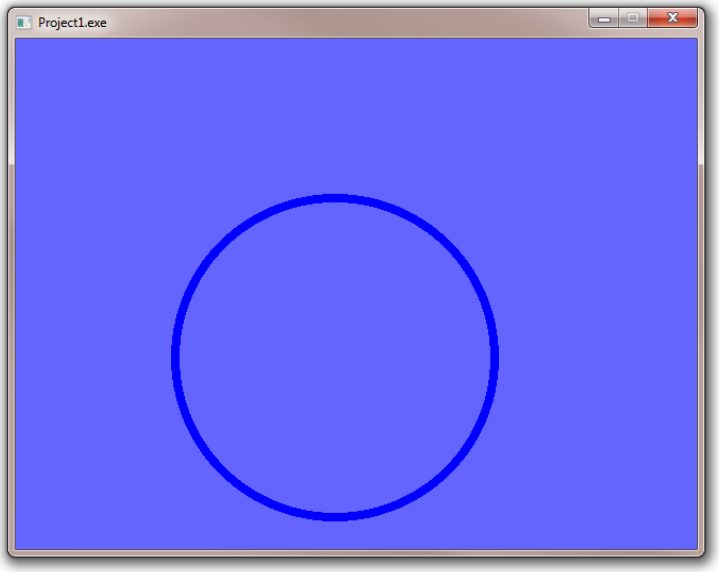
Rectangle

```
void al_draw_rectangle(float x1, float y1, float x2, float y2,
ALLEGRO_COLOR color, float thickness)
```

For example to draw a blue circle with a radius of 100 pixels and line thickness of 8 on screen at coordinates 300,300 we would type something like:

```
al_draw_circle(300, 300, 150, al_map_rgb(0,0,255), 8);
```

Producing the following:



A full list of primitives provided by Allegro can be found here:

https://www.allegro.cc/manual/5/primitives.html

You might want to consider using a layout similar to this for primitive functions:

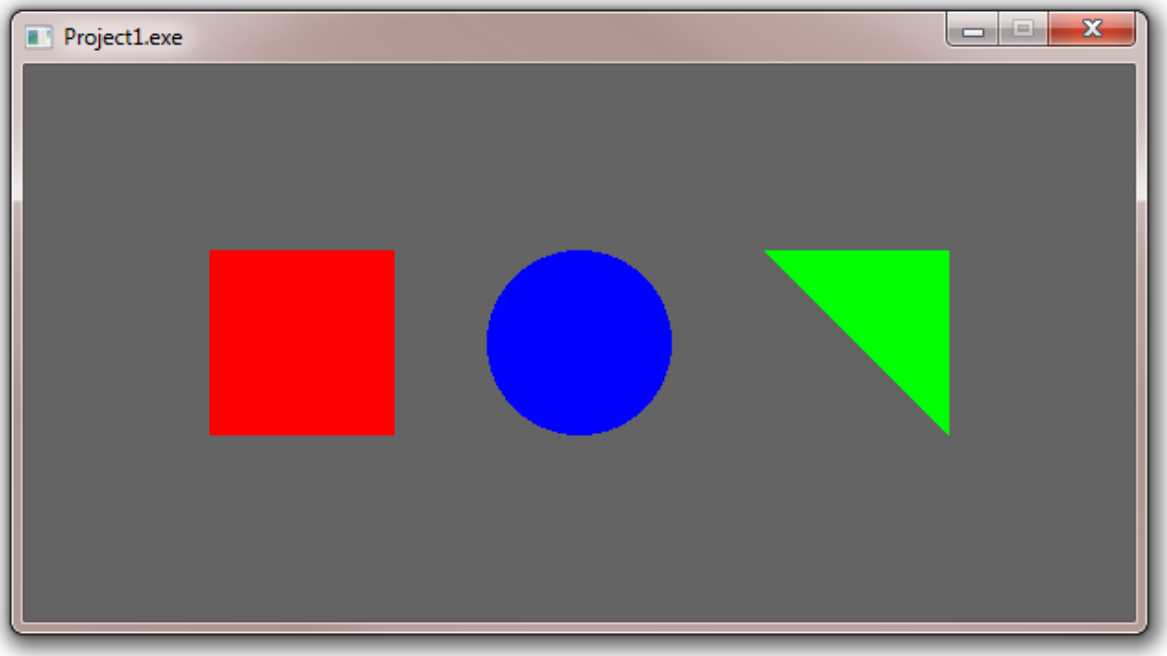
```
int squarePos[2] = {200,300};  //store x,y coords for square

int squareSize = 100;

al_draw_filled_rectangle(squarePos[0],
                         squarePos[1],
                         squarePos[0] + squareSize,
                         squarePos[1] + squareSize,
                         al_map_rgb(255,0,0));
```

Exercise 1 – Filled Square, Circle, Triangle

Your task for this exercise is to draw the following image:

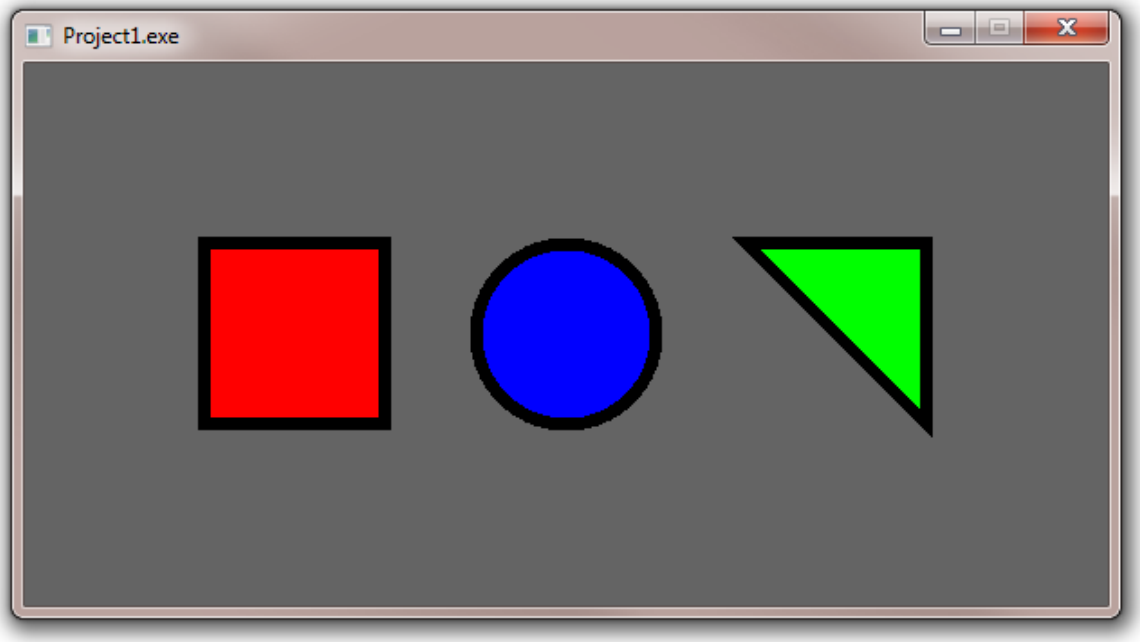


Hints

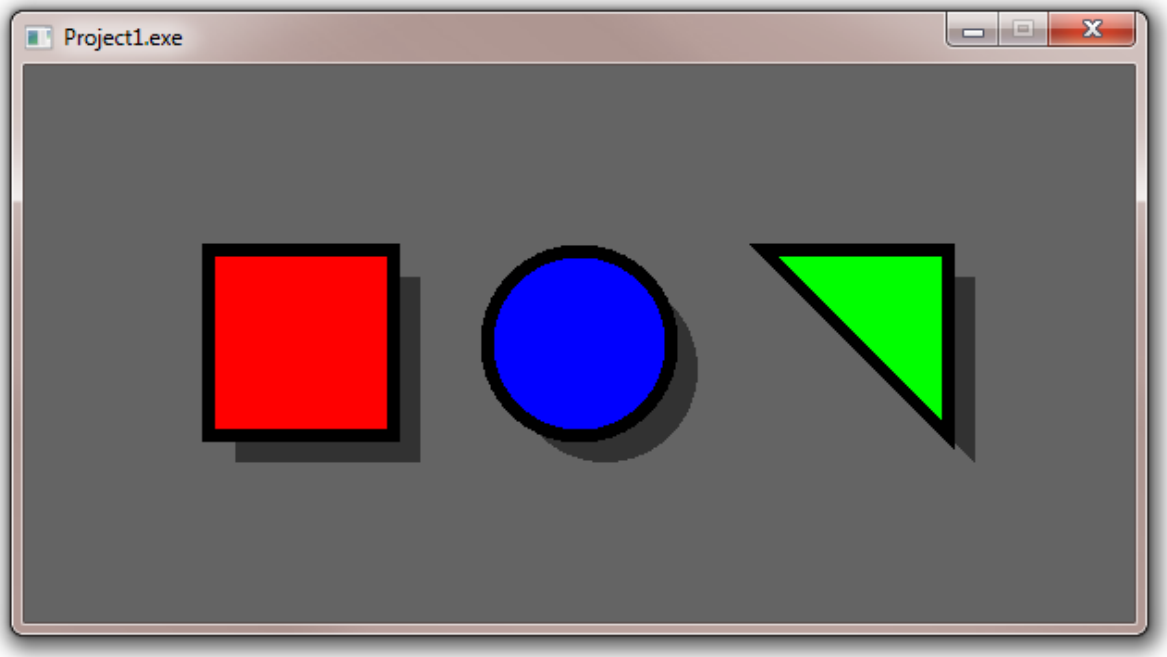
- The window has a width of 600 x 300
- The circle has a radius of 50
- Consider creating a width and height variables to store (and later access) the width and height of the allegro window
- Consider creating an int array of size 2 to store the positions of each shape
- Take a look at the Allegro API manual for a list of primitive functions

Exercise 2 – Outline and Shadow

Extend your code from Exercise 1 so that all three objects are outlined in black, like so:



Then, add a drop shadow, like this:



Hints

- This is easier than it looks
- It's mostly a case of copy-paste, although order of drawing is important

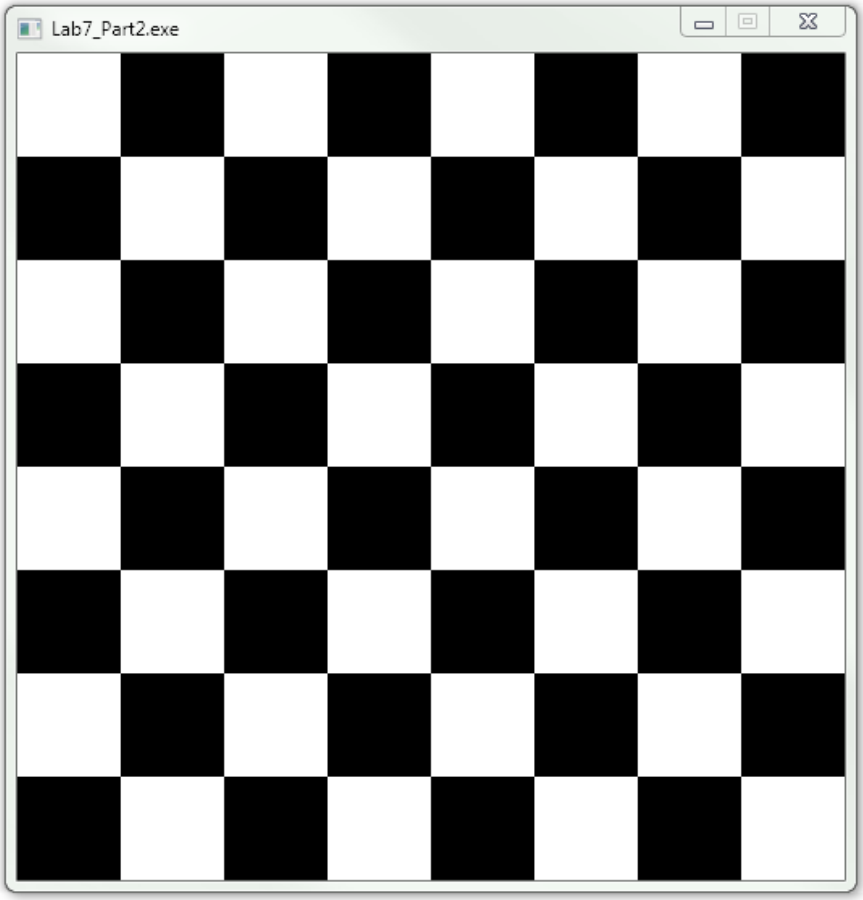
Exercise 3 – Checkerboard

Your task for this exercise is to write a program that generates a checkerboard pattern. There are many ways of doing this (google it!). Most techniques will use a nested for loop. Here's some pseudo code:

```
square_size = window_width / number_of_squares_in_row

for row=0, row < number_of_squares_in_row, row++
   for col=0, col < number_of_squares_in_row, col++
      x = square_size * col
      y = square_size * row

      if row % 2 == col % 2
            draw black rectangle (x, y, x +square_size, y +square_size)
      else
            draw white rectangle (x, y, x +square_size, y +square_size)
```

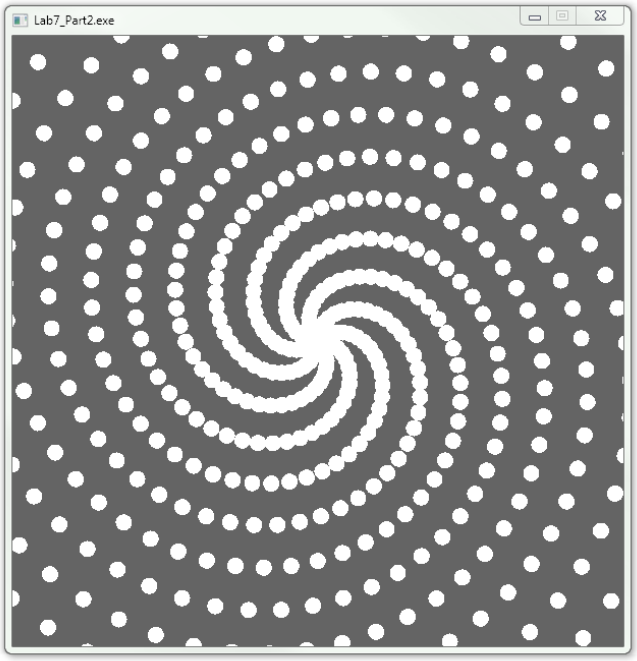


Challenge – Dot Spiral

We can describe the x and y screen coordinates of a spiral pattern using the following pseudo code:

```
for i number of points:
    angle = k * i
    float x = (1 + angle) * cos(angle * g);
    float y = (1 + angle) * sin (angle * g);
```

Where k is the (degree) change in angle between each step (e.g. 0.5), and g the tightness (e.g 0.8). Use this pseudo code to generate a spiral pattern similar to the following:



The code for generating the spiral should be placed inside a function called draw_dot_spiral which has the following prototype:

```
void draw_dot_spiral(float cX, float cY, ALLEGRO_COLOR color);
```

Where cX and cY are the coordinates of the spiral (center) and color is the spiral's colour. The function should be called similar to this:

```
//draw a red spiral centered at x=256, y=256)
draw_dot_spiral(256,256,al_map_rgb(255,0,0));
```

Hints

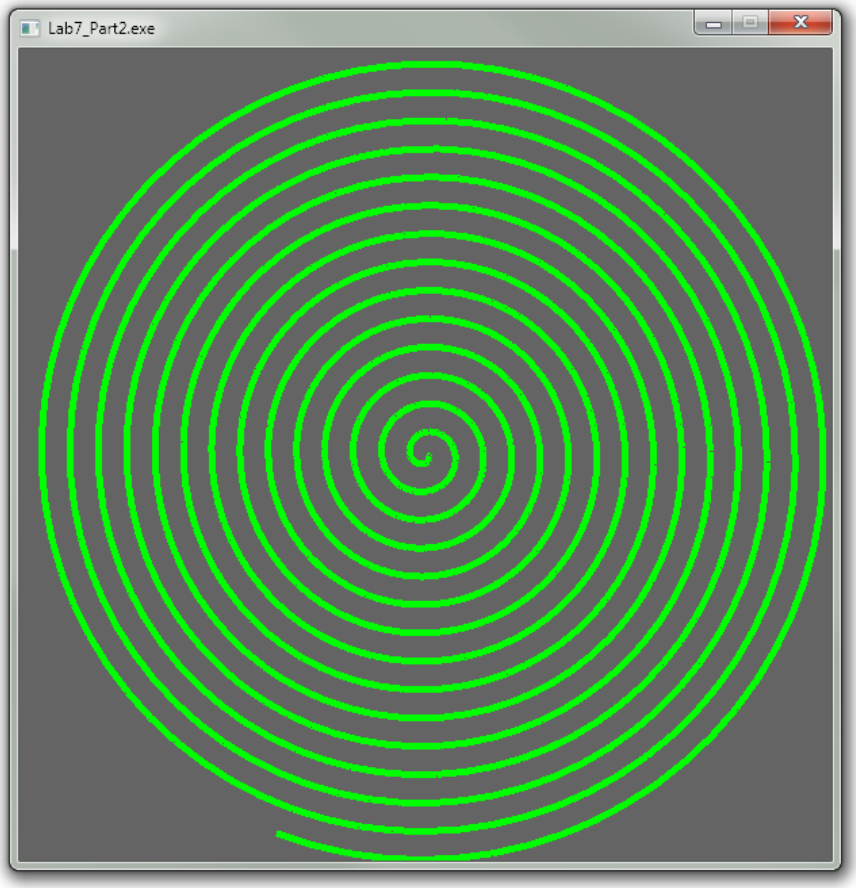
- sin and cos functions are found in **<math.h>**
- The algorithm as described in the pseudo code starts at the center and spirals outwards
- The mid-point of the screen is at width/2 and height/2.

Challenge – Line Spiral

In Allegro, we can use the al_draw_line function to draw a line:

```
void al_draw_line(float x1, float y1, float x2, float y2,
                  ALLEGRO_COLOR color, float thickness)
```

Extend your program from exercise 4 to create a spiral out of lines. Your output should resemble the following:

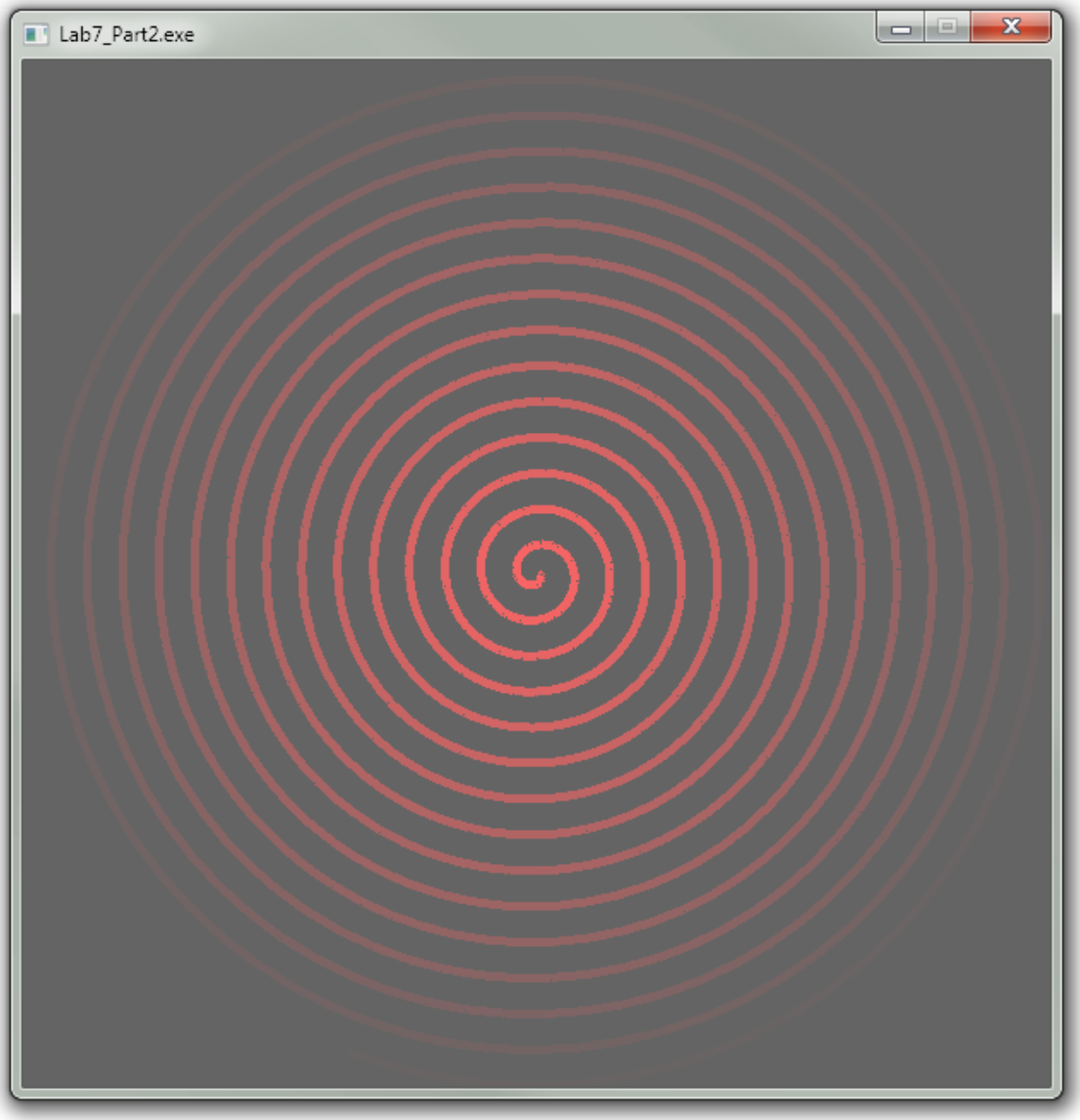


Hints

- Use a function name like draw_line_spiral() for this exercise
- Consider the dots in the previous exercise to be coordinates along the spiral
- As the line function requires four coordinates to draw a line (from x1,y1 to x2,y2) you will need to remember the spiral position calculated in the last iteration (e.g. prevX = x, prevY = y)

Challenge - Fading Spiral

Extend your spiral generator so that you can specify a start and end colour. The program should fade the spiral between these two values. For example here we are fading between a redish colour (255,100,100) and the background colour (100,100,100):



Your function declaration should look something like this:

```
void draw_line_spiral(float cX, float cY, ALLEGRO_COLOR
startColor, ALLEGRO_COLOR endColor);
```

Hints

- If there are 500 points in the spiral, 1 will be in the center, 500 on the outside
- You can find where you are in the spiral pattern as a percentage (from 0.0 to 1.0) by dividing the current iteration i with the total number of iterations. Use this to modify the colour at each iteration.

Challenge - Multiple Spirals

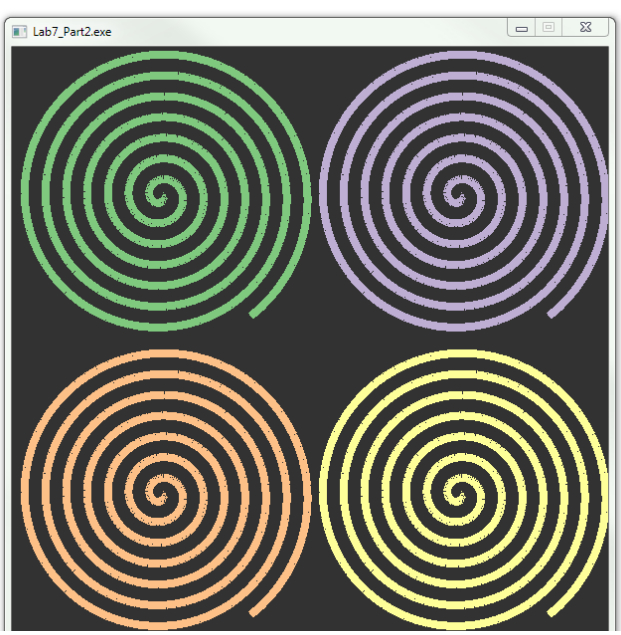
By this stage you should have a function capable of drawing a spiral between two colours. Your task for this challenge exercise is to expand your draw_line_spiral function so that you can specify all of the spiral's features by passing parameters. For example it should be straight forward to also allow customisation of the following:

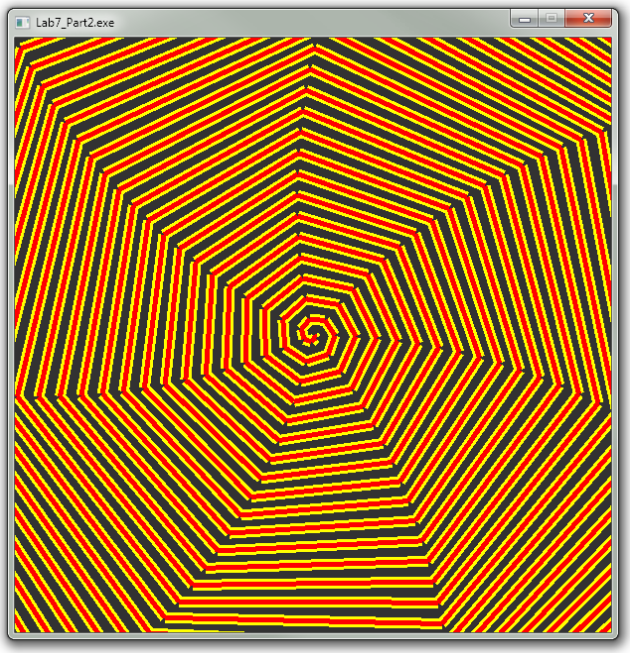
- the number of points in the spiral
- the angle between points
- the tightness of the spiral
- the thickness of the lines

Example function prototype:

```
void draw_line_spiral(float cX, float cY, int numPoints, float angleBetweenPoints,
float tightness, float lineThickness, ALLEGRO_COLOR startColor, ALLEGRO_COLOR
endColor);
```

Can you reproduce the following patterns?





Appendix A

```
#include <stdio.h>
#include <allegro5/allegro.h>

int main(int argc, char **argv)
{
   ALLEGRO_DISPLAY *display = NULL;

   if(!al_init())
   {
      fprintf(stderr, "failed to initialize allegro!\n" );
      return -1;
    }

   display = al_create_display(640, 480);

   if(!display) {
      fprintf(stderr, "failed to create display!\n" );
      return -1;
   }

   al_clear_to_color(al_map_rgb(0,0,0));

   al_flip_display();

   al_rest(5.0);

   al_destroy_display(display);

   return 0;
}
```

Appendix B

Code Snippets – Updated with example C code

Allegro 5.0 reference manual

Online C Programming Resources

Complete C Reference Library

C Programming IDE's

Dev-C++ (Windows)

Code::Blocks (Windows, Mac, Linux)

Visual Studio/C++ Express (Windows)

Netbeans C/C++ (Windows, Mac, Linux)

Codelite (Windows, Mac, Linux)

PRIFYSGOL
BANGOR
UNIVERSITY