Functions II

1. Content

2. Objective

At the end of these exercises, you should be able to:

- Explain the difference between pass by reference and pass by value.
- Explain when to use const parameters
- Do operations with a 2D vector
- Draw an image on the window
- Draw text on the window
- Know what they are and how to use recursive functions

We advise you to **make your own summary of topics** that are new to you.

3. Exercises

Your name, first name and group should be mentioned at the top of each cpp file.

Give your **projects** the same **name** as mentioned in the title of the exercise, e.g. **FunctionsBasics**. Other names will be rejected.

3.1. FunctionsBasics

Create a new project with name **FunctionsBasics** in your **1DAExx_07_name_firstname** folder.

Add your name and group as comment at the top of the cpp file.

3.1.1. Default parameters

Take the print code from last assignment Functions_1 and change the Print function declarations, turn the delimiter into a default parameter with value ` ` (a space surrounded by single quotes).

Then call these functions without a second argument. Verify the output, a space should be used.

This is a link to Default parameters

3.1.2. Pass by reference

a. CalcCosSin function

Declare and define one function **CalcCosSin** that calculates the sine and cosine of an angle. Make the sine and cosine results part of the parameter list.

Declare and define a function **TestCalcCosSin**. In this function:

- Print an info message
- Call several times the CalcCosSin function to print the sine and cosine value of 10 random angles between 0 and 2 * PI, you can use the GetRand function from a previous exercise.

```
-- Function that calculates cosine and sine --
   Radians       Cos       Sin
      0.41      0.92      0.40
      2.26     -0.64      0.77
      0.44      0.90      0.43
      0.82      0.68      0.73
      2.99     -0.99      0.15
      6.28      1.00     -0.00
      1.56      0.01      1.00
      4.24     -0.46     -0.89
      5.44      0.67     -0.75
      5.62      0.79     -0.62
```

3.2. GameFunctions

Create a new SDL project with name **GameFunctions** in your **1DAExx_07_name_firstname** folder.

Delete the generated file **GameFunctions.cpp.**

Adapt the window title.

Now that you know how to define your own functions, let's use this knowledge and define functions that solve some common game problems. As these are common problems, put also these declarations and definitions in utils.h and utils.cpp down in this area of the file:

#pragma region CollisionFunctionality

and test them in Game.cpp.

At the end of these exercises we have a utils.h and utils.cpp with following functions added to it in this area of the file:

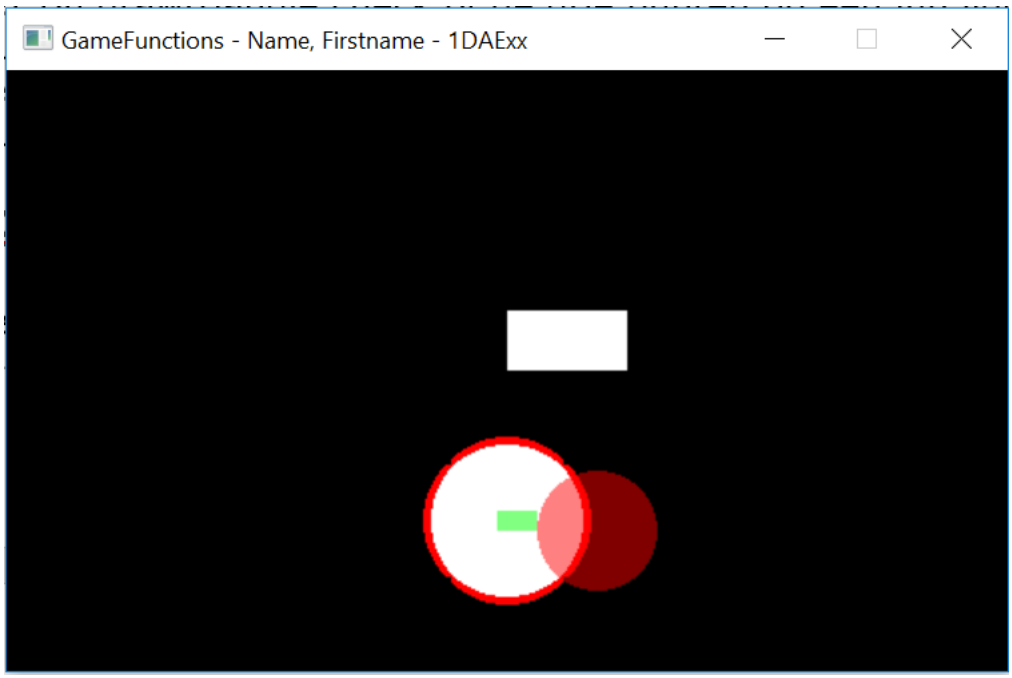
Notice that we added the **GetDistance** functions from a previous lab as well as we needed it in the other functions.

```
float GetDistance(                          );
float GetDistance(                          );

bool IsPointInCircle(                             );
bool IsPointInRect(                           );

bool IsOverlapping(                          );
bool IsOverlapping(                               );
```

And a window that looks like this.



3.2.1. IsPointInCircle

a. Circlef struct

Notice that structs.h contains the definition of Circlef. Use this Circlef struct to hold a circle's properties in the following exercises.

```
struct Circlef
{
    Point2f center;
    float radius;
};
```

b. Declaration and definition

Define a function **IsPointInCircle** which determines whether a point is within a circle's boundaries. Again, decide yourself about the necessary parameters: what information does this function need.

Tip: A point is in a circle when the distance between this point and the center is not larger than the radius. You might define a function **GetDistance** as well.

c. Test

Use this function to draw a border round a circle when the mouse enters a circle.

- Handle the SDL_MOUSEMOTION event to keep track of the current mouse position,
- Draw a filled white circle at some random position but completely situated within the window boundaries.

- When the mouse is in this circle it gets a red border, when it is outside the circle it has no border.

3.2.2. IsPointInRect

a. Declaration and definition

Define a function **IsPointInRect** which determines whether a point is within a rectangle's boundaries.

b. FillRect functions

In this test we'll draw filled rectangles.

c. Test

- Draw a filled white rectangle at some random position, but completely situated within the window boundaries.

- When the mouse enters this rectangle, it gets a red border which disappears again when the mouse leaves the rectangle.

d. Namespaces

You might have noticed that when you call a FillRect function, another FillRect declaration shows up in the intellisense. So somewhere else a FillRect function is defined. Luckily this didn't give us any problems as it has another parameter list. However, it is better to avoid naming collision by using namespaces. More info at Namespaces.

Make sure you have put your function declarations and definitions inside the namespace **utils in utils.h and cpp**. When calling them, tell the compiler to look in that namespace by preceding the call with the utils scope operator `utils::FillRect(` . Now the other FillRect function no longer appears in the intellisense.

3.2.3. IsOverlapping

a. Declaration and definition

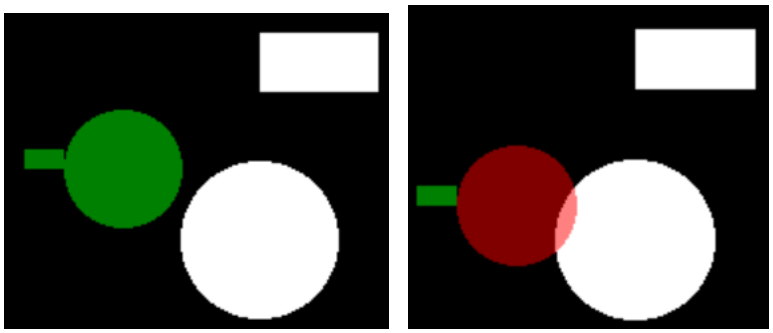
Make two **IsOverlapping** functions using overloading to determine whether:

1. Two rectangles are overlapping (more info at Rectangles overlap)
2. Two circles are overlapping

b. Test

Draw a filled semi-transparent rectangle at the mouse position (right bottom corresponds with the mouse position). When this mouse-rectangle overlaps the white one, the first one gets a red color otherwise the color is green.

Draw a filled semi-transparent circle at the mouse position (to the right of the mouse). When this mouse-circle overlaps the white circle it gets a red color otherwise the color is green.



3.3. MathVector

Create a new SDL project with name **MathVector** in your **1DAExx_07_name_firstname** folder.

Delete the generated file **MathVector.cpp.**

Copy and add your updated utils files as well.

Adapt the window title.

In math, a vector has 2 properties: a magnitude and a direction. For more info see Vectors. They are used to represent Physics types that have those 2 properties such as a force, a velocity …

To do calculations with vectors, the vector is broken up into an x and y part. The x part is the cosine of the direction times the magnitude and the y part is the sine of the direction times the magnitude.

Vector are widely used in game programming. This is an introduction to working with vectors. Later, operator overloading will make math operations with vectors more intuitive.

3.3.1. The Vector2f struct

The 2 stands for a 2D vector, we have no z-part and the f indicates that the components x and y are float types.

Add the Vector2f struct definition to structs.h that is part of the framework.

```
struct Vector2f
{
    float x;
    float y;
};
```

3.3.2. Vector functions

Declare and define following vector functions in the utils.h and utils.cpp files of the framework. Test them in Game.cpp of this project.

In the end you should have these math vector functions.

```
Vector2f Add(                                    );
Vector2f Subtract(                                 );
float DotProduct(                                 );
float CrossProduct(                                 );
std::string ToString(                 );
Vector2f Scale(                              );
float Length(                      );
Vector2f Normalize(                    );
float AngleBetween(                                 );
bool AreEqual(                                      );
```

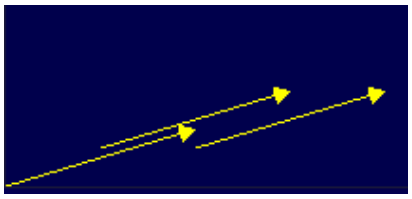
And a function that Draws a vector.

```
void DrawVector(
```

a. DrawVector

This function has 2 parameters: the vector itself (Vector2f) and the starting point (Point2f). It draws the vector as a line between the starting point and the position as indicated by the x and y properties of the Vector2f parameter. Draw an arrow at the end point (angles are 30 degrees) as a filled triangle. You can use the atan2 function (The atan2 function) to get the angle between the vector and the horizontal x- axis. Make the starting point parameter optional, when omitted the starting point should be the origin.

Use this function to draw the same vector at different start positions.



b. ToString

Returns a string that produces this result: "[x, y]" where x and y are the values contained in the vector. The function does not print!

Test the function in Start(): call this function using the previous vector as an argument and send the result of this function to the console. You will need to copy the vector definition in the Start function.

```
[100.000000,  30.000000]
```

c. Add

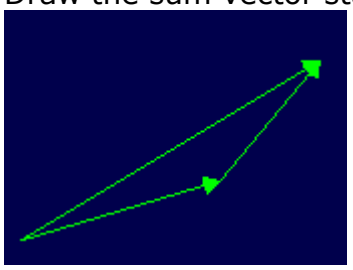
Returns the sum of two vectors.

Test it in following way:

Draw the first vector starting at some point on the window.

Draw the second vector starting at the end point of the first vector.

Draw the sum vector starting at the start point of the first vector.



d. Subtract

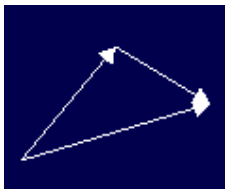
Returns the difference between the first (v1) and second vector (v2): v1 - v2

Test it in the **Start()** function the following way:

Draw the first vector starting at some point on the window.

Draw the second vector starting at that same point.

Draw the difference vector starting at the end point of the second vector.



e. DotProduct

Dot product

Returns the dot product of two vectors.

Test it in the **Start()** function as follows:

- Define 2 orthogonal vectors.
- Then calculate the dot product, it should be equal to 0.
- Print the two vectors and their dot product
- Define 2 vectors that are parallel to each other.
- Then calculate the dot product, it should be equal to the product of the lengths of both vectors.
- Print the two vectors and their dot product

```
Dot product:
hor = [80.000000,  0.000000]    ver = [0.000000,  50.000000]    Dot product: 0.000000
ver = [0.000000,  50.000000]    para = [0.000000,  25.000000]   Dot product: 1250.000000
```

f. CrossProduct

Cross product

Returns the length of the Z_component of the cross product of two vectors.

Test it in the **Start()** function as follows:

- Use the same vectors as in the dot product: hor and ver
- Check that the cross product of hor and ver returns a positive value and of ver and hor a negative one.

```
Cross product:
hor = [80.000000,  0.000000]    ver = [0.000000,  50.000000]    Cross product: 4000.000000
ver = [80.000000,  0.000000]    hor = [0.000000,  50.000000]    Cross product: -4000.000000
```

g. Length

Returns the length of a vector.

Test it in the **Start()** function as follows:

- Call it for a vector and verify that the returned value is ok.

```
Length of [100.000000,  30.000000] is 104.403069
```

h. Scale

Returns a new vector that is the result of multiplication of a vector and a scalar.

Test it in the **Start()** function as follows:

- Test it by scaling previous vector by 0.1.

```
Scaled [100.000000,  30.000000] is [10.000000,  3.000000]
```

i. Normalize

Returns a vector that has the same direction as the argument, but its length is 1.

Test it in the **Start()** function as follows:

- Call it for previous vector and verify that the length of the returned vector is 1.

```
Normalized [10.000000,  3.000000] is [0.957826,  0.287348] its length is: 1.000000
```

j. AngleBetween

Returns the angle between two vectors. There are several ways to calculate this angle however as a programmer you should prefer a method that doesn't call the sqrt function. **Inverse tangent of cross/dot** gives the angle

angle = atan2(cross, dot)

More info about the atan2 function.

Test it in the **Start()** function as follows:

- Verify the correct working by getting the angle of some vectors, e.g. for the angle between a vector along the x and the other along y axes.

```
Angle between [10.000000,  0.000000] and [10.000000,  10.000000] is: 0.785398 radians, or 45.00
0000 degrees.
```

k. AreEqual

Returns true if two vectors are (almost) equal. Small errors can occur when two floating point variables are compared using the == operator. Never use == on floating point variables. Instead, see how small the difference between the components of both vectors is. A good value is 0.001f.

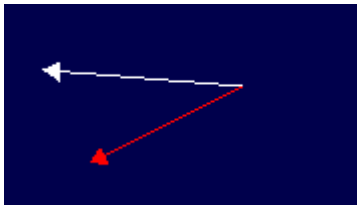
Check, but no printing is required.

l. Animating projection

In graphics programming, the scalar projection of two vectors is often needed. This is the projection of one vector onto another. The other one needs to be a normalized vector. This is e.g. used to reflect an object when colliding with a wall.

Make the following in the Draw function:

- Define and draw a vector that rotates with a framerate independent rotational velocity.
- Define a normalized vector that is not horizontal, nor vertical.
- Calculate the dot product.
- Multiply the normalized vector with the dot product. (= the projection)
- Draw that resulting vector in red.



3.4. Recursion

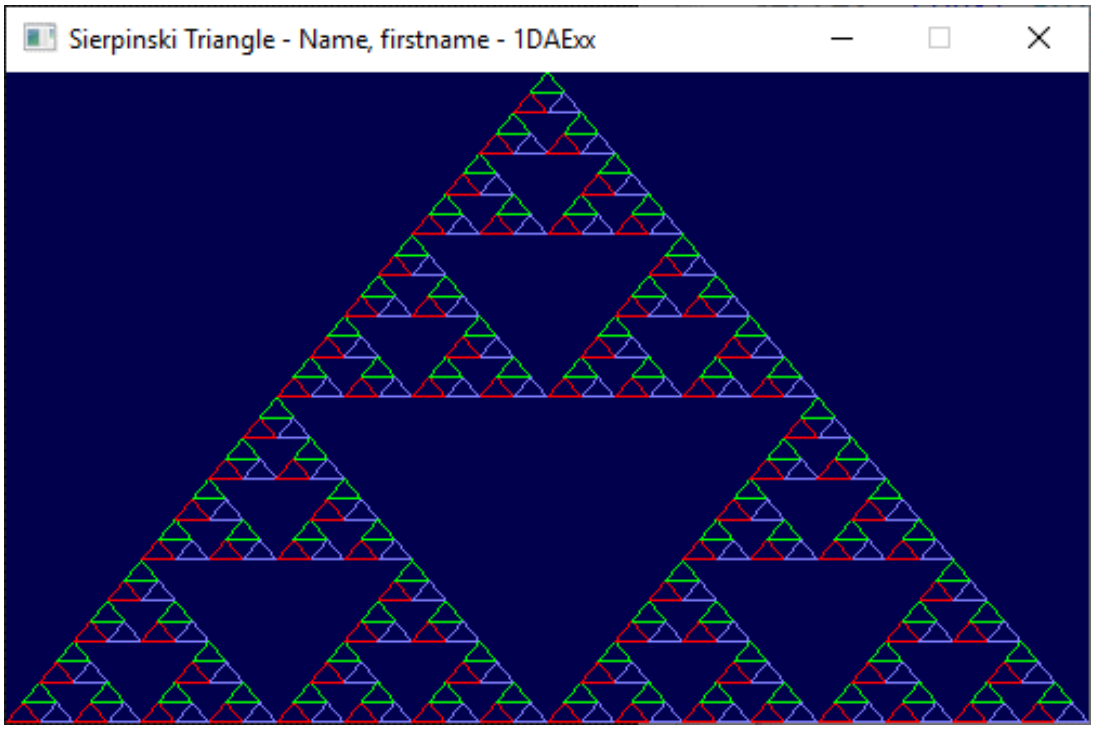
Create a new project with name **Recursion** in your **1DAExx_07_name_firstname** folder.

Delete the generated file **Recursion.cpp.**

Adapt the window title.

While making this exercise, decide yourself if you need to copy and add your utils files.

In this project you'll make an exercise on Recursion. Recursion can help in displaying complex patterns where the pattern appears inside itself as a smaller version. Such patterns, called **fractals** are in fact a visual representation of the concept of recursion. After making this exercise you should have a window that showing the Sierpinski triangle

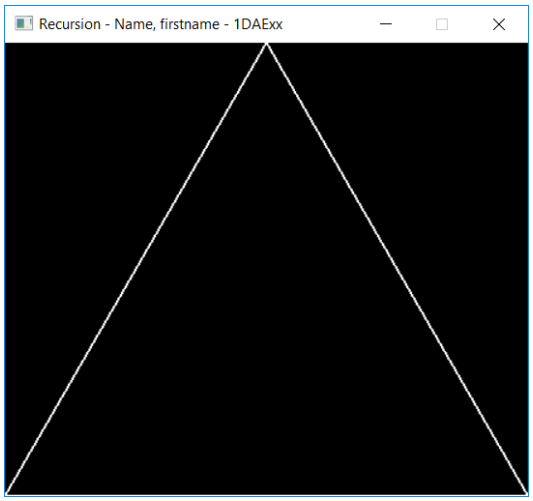


a. Function that draws a triangle

Declare and define a function that draws a triangle using the given vertices.

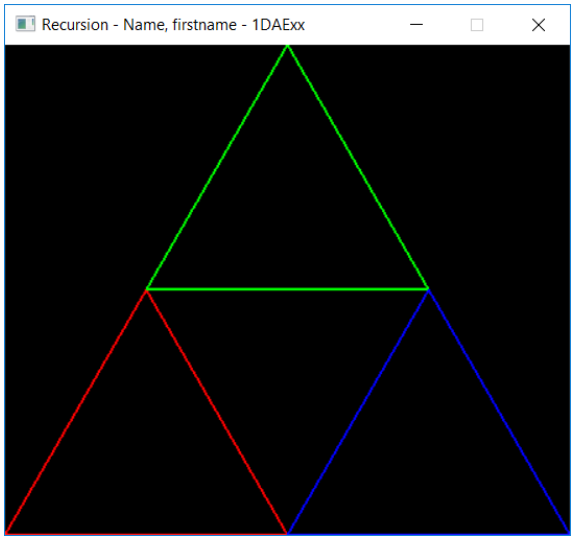
```
void DrawSierpinskiTriangle( const Point2f& left, const Point2f& top, const Point2f& right );
```

And call it from the Draw function, so that you get this result.



b. Function that draws one triangle and 3 inner rectangles

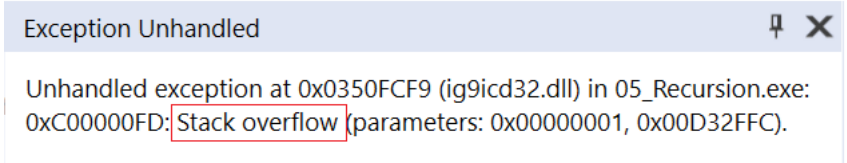
Change the function. After having drawn the main triangle, draw 3 other triangles inside it and using another color for each one. Like this.



c. Recursive function

Now you have a function that first draws a triangle and then calls three times itself to draw 3 inner triangles.

Rebuild and test. An exception occurs because a function calls itself without ever stopping which causes a stack overflow.



d. Recursive function that stops calling itself

Solve this problem. Add a condition that stops the recursive call when the distance between the given left and right vertex is less than 20.

3.5. ImageAndText

There is a video tutorial demonstrating how to work with images in an SDL project.

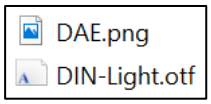
Create a new project with name **ImageAndText** in your **1DAExx_07_name_firstname** folder.

Delete the generated file **ImageAndText.cpp.**

Adapt the window title.

Building this project should give no errors.

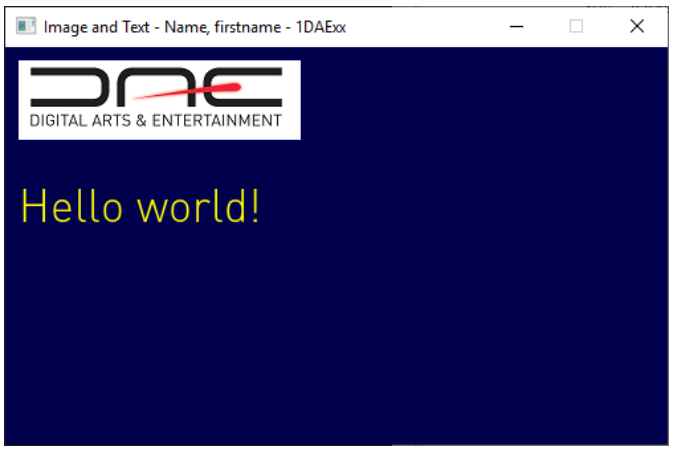
Create a folder **Resources** in this project folder and copy the given resources into it.



In the video you'll see how to:

- Load a Texture from a png file
- Load a Texture from a text you want to render in a given font and color
- Delete the created resources

At the end of this exercise you should have a window that shows an image and some text.



3.6. FreeGame

Continue improving your game of previous week.

When the i-key is pressed then print some information on the console:

- Short description of the game
- Which keys to use
- …

4. Submission instructions

You have to upload the folder *1DAExx_07_name_firstname, however first clean up each project. Perform the steps below for each project in this folder:*

- In Solution Explorer: Select the solution, RMB, choose **Clean Solution**.
- Then **close** the project in Visual Studio.
- Delete the .vs folder.

Compress this *1DAExx_07_name_firstname* folder and upload it before the start of the first lab next week.

5. References

5.1. Functions

5.1.1. Introduction

http://www.learncpp.com/cpp-tutorial/14-a-first-look-at-functions/

http://www.learncpp.com/cpp-tutorial/1-4a-a-first-look-at-function-parameters/

http://www.learncpp.com/cpp-tutorial/1-4b-why-functions-are-useful-and-how-to-use-them-effectively/

http://www.learncpp.com/cpp-tutorial/1-4c-a-first-look-at-local-scope/

5.1.2. Function overloading

http://www.learncpp.com/cpp-tutorial/76-function-overloading/

5.1.3. Default parameters

http://www.learncpp.com/cpp-tutorial/77-default-parameters/

5.1.4. Forward Function declaration

http://www.learncpp.com/cpp-tutorial/17-forward-declarations/

5.2. Rectangles overlap

http://www.geeksforgeeks.org/find-two-rectangles-overlap/

5.3. Vectors

5.3.1. Vector and some basic operations

https://www.mathsisfun.com/algebra/vectors.html

5.3.2. Dot product

https://www.mathsisfun.com/algebra/vectors-dot-product.html

5.3.3. Cross product

https://www.mathsisfun.com/algebra/vectors-cross-product.html

5.3.4. Visual explanation dot and cross product

https://www.youtube.com/watch?v=h0NJK4mEIJU

5.3.5. The atan2 function

http://www.cplusplus.com/reference/cmath/atan2/

5.3.6. Scalar projection

https://en.wikipedia.org/wiki/Scalar_projection