Variables part 2

1. Content

2. Objective

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

- Explain what the Visual Studio build does and how to solve the reported errors
- Define, initialize and manipulate the std::string type
- Convert a string containing a number to a numeric value and vice versa
- Define and use simple functions
- Use existing functions that calculate a result: cos, sin, sqrt…
- Find and solve errors and warnings related to the above topics
- Generate random numbers in given ranges
- Get familiar with the modulo operator
- Typecast types to other types, the integer division
- Explain the difference between the prefix an postfix increment/decrement
- Avoid magic/hard coded numbers that make your code less readable and hard to maintain
- Draw using transparent colors

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

3. Exercises

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

```
// Name, Firstname - 1DAEXX
```

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

3.1. Functions

Create a new project with name **Functions** in your **1DAExx_02_name_firstname** folder.

In the following exercises, you will learn how to define your own simple functions and how to use (call) existing functions.

3.1.1. Define your own functions

Remove the generated file Functions.cpp (choose delete) and copy the given **02_FunctionsStart.cpp** file into the folder of this project. Rename it into **Functions.cpp**.

Then add this Functions.cpp file to the **Source Files** filter in the **Solution Explorer** of **Visual Studio** (Select this filter, LMB (left mouse button) and select "**Add existing item**").

The demo in the theory lesson showed you:

- What a function is in its most basic form, and why we will already use it now.
- How to split a large block of code into smaller **functions.**
- How to execute the code inside the function; we speak about **calling a function.**
- What happened after the function was called.
- What the function definition and declaration is.

We expect that from now on you always avoid having long functions by splitting them up in smaller ones. It makes your code better, more readable. See this reference: C++ Coding standards Rule 20: Avoid long functions

3.1.2. Using existing functions that have one parameter

While programming you don't want to be bothered by having to write your own code to calculate trigonometry, I assume? Luckily there are a huge number of already made math and other **functions** at our disposal and ready to be used in our games. In this part we will build up some experience by using them.

Put the code of following exercise in a function **TestOneParFunctions.**

Just like on your calculator also in C++ Math functions are available, such as: **cos**, **sin**, **tan**, **acos**, **asin**, **atan**, **sqrt**, **abs**, **fabs**…

In console applications (without sdl), you will need to include add this before you can use the trigonometry functions or M_PI, below the #include "pch.h" line:

```
#define _USE_MATH_DEFINES
#include <cmath>
```

To use them you need to enter the name of the function followed by a pair of parentheses, that enclose the extra information that the function needs, e.g. the angle in radians in case of the cos, sin, tan… functions. These pieces of extra information are called the **parameters** of the function. You can pass this information – these are called the **arguments** - in several forms as long as it suits the required type:

- a literal constant
- a variable
- an expression
- …

These functions have a **result** of a **specific type**. Calling such a function is also an expression and can be used in other expressions, can be assigned to a variable, can be sent to the console…

These trigonometry functions are made to have a **double** parameter. There are **float** versions too: **sinf, cosf**, etc…So if you use float parameters, make sure to use these versions.

Let's do some exercises.

- Call the cos function using a **literal as argument**. Be aware C++ math functions use **radians**, not angles!
- Define and initialize a variable *angle to 0* and call the cos function using this **variable as argument**.
- Calculate the cos value of next angle values: 0, 1*π/4, 2*π/4, 3*π/4, π radians. Use the *angle* variable that you increment with the value π/4. Print results on the console. Don't use any for loops yet.
  Create a pi variable with the value 3.1415926535.

```
Use existing functions with 1 parameter
Angle: 0
  cos: 1
Angle: 0.785375
  cos: 0.707123
Angle: 1.57075
  cos: 4.63268e-05
Angle: 2.35613
  cos: -0.707058
Angle: 3.1415
  cos: -1
```

- You can also use the **result of an expression as argument** in a function call. Call the cosine function passing the expression **angle * 2**.

3.1.3. Using existing functions that have more parameters

Put the code of this exercise in a function **TestTwoParFunctions.**

Above functions only require one parameter; some need more pieces of information though. Then you must separate them by commas and follow the prescribed sequence.

The **pow** function has 2 parameters. The function computes a base number raised to the power of exponent number. This function is defined in <cmath> header file. The return type of the function is a double, what can be unexpected. So, make sure to store the result in a double, or typecast to an integer. See also: http://www.cplusplus.com/reference/cmath/pow/

Example:

```
double result = pow(2, 8);
```

Add this calculation using variables, not hard coded numbers and print the result.

Pythagoras Theorem:

More information here: https://www.mathsisfun.com/pythagoras.html

Calculate the longest side (hypotenuse) of the triangle where the short sides are 3 and 4. The result should be…5. Print the result.

You will need the functions **pow** and **sqrt.**

3.2. Variables basics 2

Use Visual Studio to create a new **project** with name **Variables2Basics** in your **1DAExx_02_name_firstname** folder.

Here we will explore the modulo operator and how to generate random numbers.

3.2.1. The modulo operator %

Watch the slides video about this modulo operator or read on this site: learncpp.com, then make these exercises.

In short, the modulo operator results in the remainder after division.

e.g. 5%4 -> 1

Ask the user for a number and use the modulo operator to print a 0 if the number is even, and print 1 if the number is odd. Do this without if-else structures (that you are not supposed to know yet).

3.2.2. The rand function

http://www.cplusplus.com/reference/cstdlib/rand/

This function generates pseudo random numbers between 0 and RAND_MAX. The latter is guaranteed to be at least 32767.

3.2.3. Modulo % and random numbers

The modulo operator is commonly used in combination with the rand() function: Generating random numbers.

At the start of a console application, initialize the random generator using srand, see Initialize random number generator. Hereby use **nullptr** (since C++11) as time argument instead of NULL or 0 (used in older C++ versions and still used in many examples on the internet): `srand( int ( time( nullptr ) ) );`

For framework application this srand already happens in the main function.

Now write code that generates a random integer number in the following inclusive intervals:

- [0, 50]
- [10, 80]
- [-20, +20]
- [-2, +2]

Then write code that generates a random floating point (!) number in the following inclusive intervals and with rounding to 2 decimal places:

- [5.00, 10.00]
- [-5.00, +5.00]

3.2.4. Typecasting

When copying: a literal constant, the content of a variable, the result of an expression, the result of a function call … into a variable of another type – e.g. during an assignment operation, or when passing as a parameter value to a function - explicit typecasting is sometimes required. The Visual Studio compiler

warns about this. These warnings can be eliminated by typecasting or by using literals of the same type.

When typecasting a floating point type to an integral type, digits behind the decimal point are lost.

Make following exercises:

- Ask the user to enter a letter on the console, show this letter and its ASCII value on the console.

```
Letter? c
Letter c, ASCII value 99
```

- Generate a random letter in the interval [a, z] without using the ASCII values of a and z. Then show the random letter and its ASCII value (typecast to int).

```
Random letter v, ASCII value 118
```

- Consider 3 float values: 5.44, 5.45 and 5.51. Cast them to an int and round them using the **round** function (Function round). Show both results on the console. Notice the different results between casting and rounding.

```
5.44, rounded: 5, int cast: 5
5.45, rounded: 5, int cast: 5
5.51, rounded: 6, int cast: 5
```

- Calculate and print the aspect ratio of full Hd (1920 x 1080)

```
The aspect ratio of full HD (1920 x 1080) is: 1.77778
```

3.2.5. Prefix and postfix increment/decrement

Consider the code snippet below.

First try to figure out what will be printed on the console?

Then copy this code into your cpp file and verify whether your answer was correct.

```
int i{10};
int j{++i};
std::cout << "i: " << i << ", j: " << j << std::endl;

int k{10};
int l{k++};
std::cout << "k: " << k << ", l: " << l << std::endl;
```

3.3. Text using std::string

The std::string is a complex type that represents text. A collection of characters. It is part of the standard template library.

To be able to use the std::string, you will need to include <string>

We will now explore the string.

3.3.1. Declaration and initialization

Add a function “ExploreStrings”.

Declare and print a string without initializing it.

Declare and print a string using a text literal to initialize it. This is a string literal: "I am a string literal", including the quotes.

Change the value of the string by using the assignment operator and a string literal. Print the result.

Use the console to ask the user to enter a text (without spaces), the extract operator (>>) will copy the typed characters into the std::string. Be aware, when cin sees a space, it will stop reading chars from the keyboard buffer.

3.3.2. String concatenation (+)

The + operator is defined for string type variables. It creates a new string with a value that is the concatenation of the characters in both operands.

Make some exercises using this operator:

- Make a new string variable which gets as value the concatenation of two strings entered by the user. Show the resulting string on the console.

```
First word? key
Second word? stroke
keystroke
```

- Try several other combinations of string concatenation:
  - a string literal and a string type variable
  - a character variable and a string type variable
  - a character literal and a string type variable

3.3.3. Converting numbers to strings.

Try to concatenate an integer or a float literal/variable to a string object. Does this work? The **to_string** function comes to your rescue, look it up on the internet (Function to_string) and use it to solve this problem. Use this new knowledge to solve following exercise:

Define a string object with name *numbersLog* and **one** integer variable *enteredNr*. Ask the user 3 times to enter an integer value and each time capture the entered value in **enteredNr**. Meanwhile use string concatenation to keep a log of the 3 entered numbers in *numbersLog*. Display this log in the console.

```
Enter an integer value: 20
Enter an integer value: 30
Enter an integer value: 40
Log of entered numbers : 20 30 40
```

3.3.4. Converting strings to numbers

Define 2 **string** objects initializing them with numerical values (one containing an integer number e.g. "3", the other containing a float value e.g. "3.1415") Suppose you need to add these 2 values in a mathematical way. The + operator applied on string operands concatenates them. So, you first need to convert both strings to an integer and float. Look on the cplusplus.com website which functions get the number out of these string objects, so that you can add them with each other.

```
One string contains 3
Other string contains 3.1415
Adding strings 33.1415
Adding numbers 6.1415
```

3.4. StaticDrawing

*If you already have programming experience, you can skip the exercises that might seem repetitive and spend more time on the more challenging and the free exercise described at the end of this document. However, do the **pentagram** and **columns graph** exercises as you'll animate them in one of the later assignments.*

Be careful: do not overestimate your programming skills! This is a common student mistake.

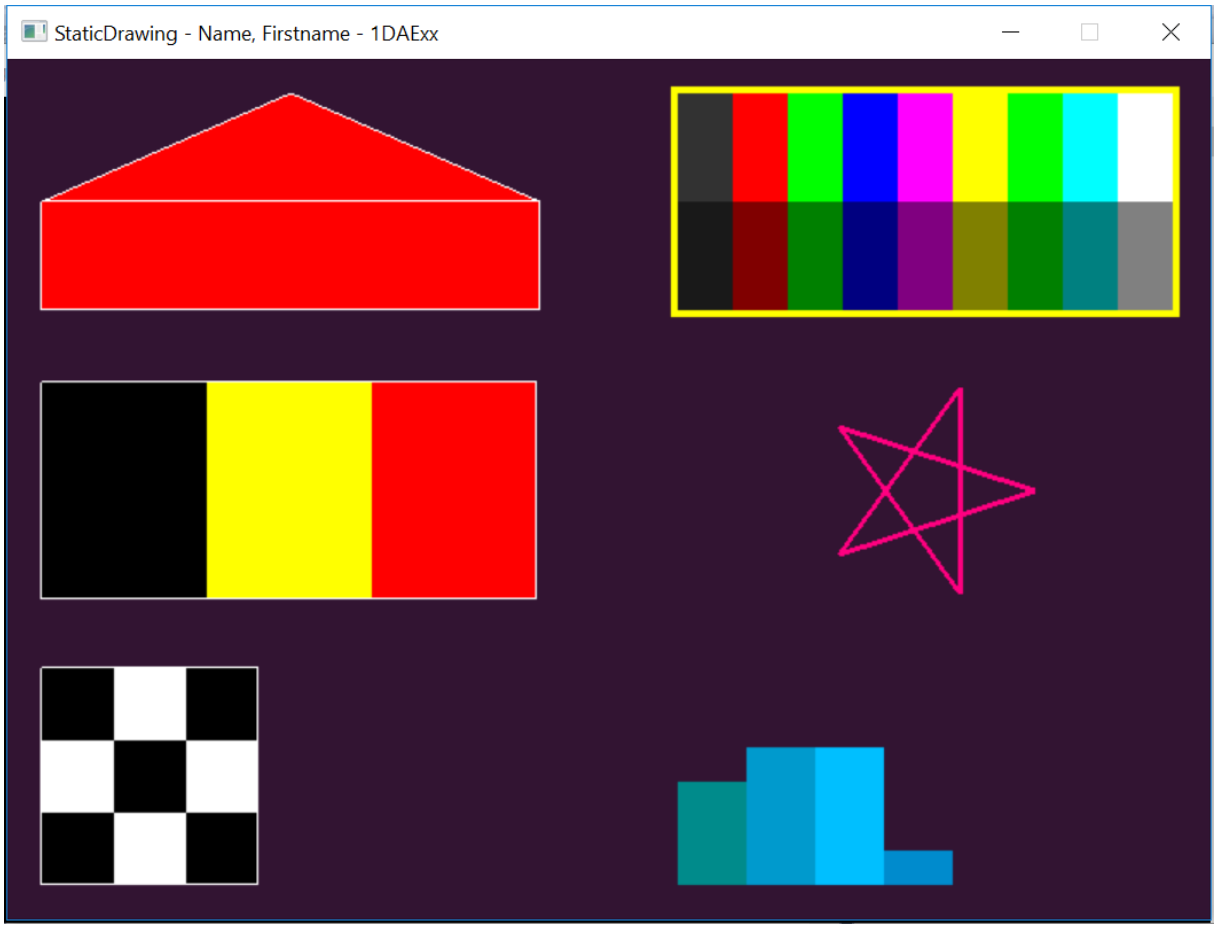
Create a new project with name **StaticDrawing** in your **1DAExx_02_name_firstname** folder.

Delete the generated **StaticDrawing.cpp** file

Use the framework files.

Adapt the **title** of the window.

Then make the exercises as described further in this document. In the end you should have a window that looks like this.



3.4.1. Make the 6 drawings

Use **variables** whenever you can, make the figures easily adjustable by using variables and expressions instead of hard coded numbers.

Make your code easy to read – for yourself and your teacher – and split it into functions. As a function performs a task use a **verb** in its name e.g.:

```
void DrawHouse( );
void DrawFlag( );
void DrawCheckerPattern( );
void DrawColorBand( );
void DrawPentagram( );
void DrawColumnChart( );
```

These functions should be declared in the Game.h header file where it is mentioned as comment. The definition is in the Game.cpp file below all the other functions.

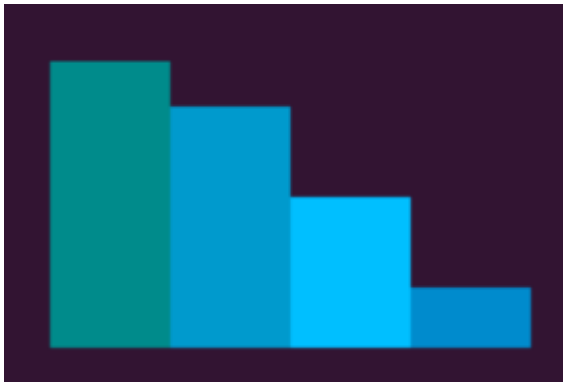
The drawing should contain following parts:

1. House (do this as the last one)
2. Belgian flag
3. Checker pattern
4. Colored band
   - Draw 9 rectangles that are 4 times higher than they are wide in different colors. Do not forget the yellow border.
   - After that, darken the lower half by drawing a filled dark rectangle using a color that has an alpha component that is less than 1.
5. Pentagram: use trigonometry to determine the coordinates of 5 points situated on a circle and then draw lines between these points.
6. Column chart of game playing percentage per following age groups:
   - [0,20]
   - [20,40]
   - [40,60]
   - Older than 60

3.4.2. User input

Change your code to let the user enter the 4 percentages of the column chart. When your code was written well, you should only have to add code that handles the user input and the drawing code should use these entered values without any changes.

```
% people playing games
In the range [0, 20]? 95
In the range [21, 40]? 80
In the range [41, 60]? 50
Older than 60 ? 20
```



3.4.3. Maintainable and readable code

Ask yourself the question what code you need to modify when you want to change following values:

1. House : position, width and height
2. Flag: position, width and height
3. Checker pattern: position and size
4. Colored band: position and width
5. Pentagram: center and radius
6. Column chart: position, column width, percentages

What changes did you need to make?

1. Only initialization of variables?
2. The drawing operations too?

When you need to change the drawing operations as well, then **adapt your code** so that changing the drawing code is no longer necessary when another position or size is required.

3.5. Random Rectangles

Create a new project with name **RandomRectangles** in your **1DAExx_02_name_firstname** folder.

Delete the generated **RandomRectangles.cpp** file. Use the framework files. Adapt the window title.

Generate a rectangle with rand color, random size and position. The rectangle must stay inside the window and keep a distance of 10 pixels from the edges. Because of the high framerate, a new rectangle will be drawn each frame. If you are sensitive to high frequency screen flashes, DO NOT MAKE THIS EXERCISE.

3.6. Beam

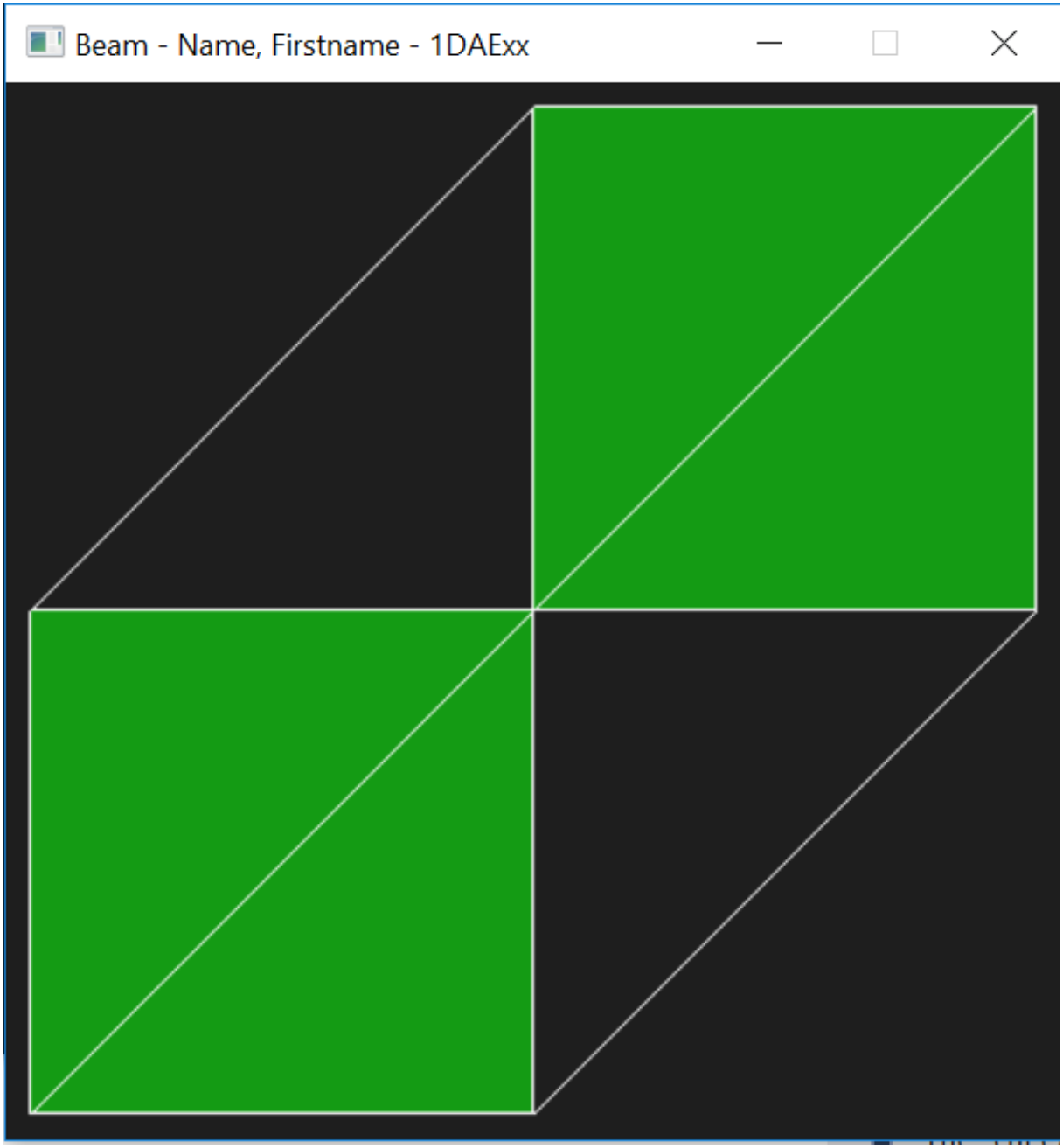
Create a new project with name **Beam** in your **1DAExx_02_name_firstname** folder.

Delete the generated **Beam.cpp** file.

Use the framework files.

Adapt the window title.

Write the code to get the image below taking into account the restrictions as listed in next paragraph.



3.6.1. Restrictions

Comply with following restrictions:

- The application window size is 420 x 420 pixels
- The background is black.
- The squares are green and have a white border and their vertices are connected with white lines.
- Use a local variable **squareSize** to hold the size of the squares, the value is 200.
- Use a local variable **border** to hold the distance to the window-border, the value is 10.
- You are allowed to use only the variables g_WindowWidth, g_WindowHeight, squareSize and border**, other** variables are **not** allowed.

3.6.2. Change the window size

Changing the window size to 550 x 450 should result in this drawing without changing any other code. Verify this in your code.

3.7. Star

Create a new project with name **Star** in your **1DAExx_02_name_firstname** folder.

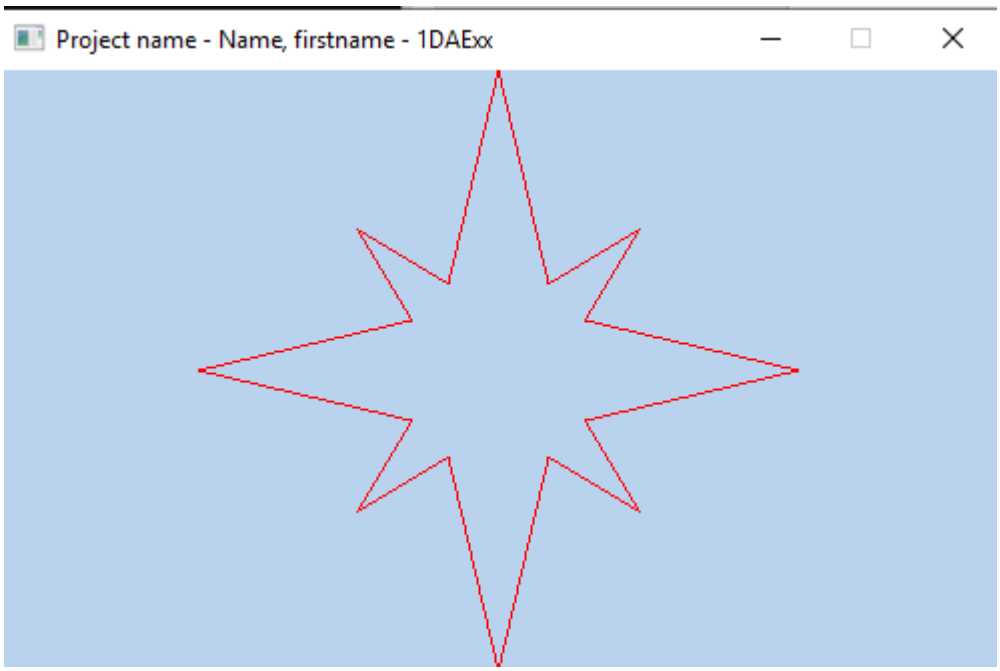
Delete the generated **Star.cpp** file.

Use the framework files.

Change the title of the window and adapt the window size into 500 x 500.

Then make the exercise as described further. In the end, you should have a window that looks like this.

Below is a more detailed description on how to proceed.



3.7.1. Use variables instead of magic numbers (C++Coding standards Rule 17)

The **outer radius** of the star is 200 pixels. Use a variable to hold this value.

Also use variables to hold the values of 2D Cartesian **coordinates of the 16 points**. It is important that you choose these variable names very carefully, because you will end up in 16*2 (x and y) variables.

It's also a good idea to first draw the star on paper and indicate on it the names you give to the 32 vertices coordinates.

3.7.2. Split a problem into smaller, easier parts

The easiest way to solve this problem is to calculate the coordinates of the vertices in 2 steps:

1. Calculate the coordinates of the vertices supposing that the star‘s center coincides with the origin of the coordinate system.
2. Translate the star center towards the center of the window.

3.7.3. Center of the star located in the origin

The vertices are located on circles with center in the origin.

- 4 vertices on a circle with a radius equals the outer radius and having an angle of 0, 90, 180 and 270 degrees with the x-axis
- 4 vertices on a circle with a radius equals 2 / 3 * the outer radius and having an angle of 45, 135, 225 and 315 degrees with the x-axis
- 4 vertices on a circle with a radius equals 1 / 3 * the outer radius and having an angle of 30, 120, 210 and 300 degrees
- 4 vertices on a circle with a radius equals 1 / 3 * the outer radius and having an angle of 60, 150, 240 and 330 degrees

3.7.4. Translate center of the star towards center of the window

Just add:

- the value g_WindowWidth / 2 to the x-values of the vertices.

- the value g_WindowHeight / 2 to the y-values of the vertices.

3.7.5. Draw the star

Draw lines connecting the points.

3.7.6. Does your code still work for another radius value?

Change the radius. If you did not use a radius variable, now is the time to do it!

Your code should still work fine without any further changes.

4. Submission instructions

- You have to upload the folder *1DAExx_02_name_firstname, however first **clean up** each project. Perform the steps below **for each project** in this folder:*
  - **Close** the Visual Studio.
  - **Remove** the **debug**, **x64** (if present) and the (hidden) **.vs** folder

Compress this *1DAExx_01_name_firstname* folder to a zip file, rename the file to contain both team names:

1DAExx_01_name1_firstname1 name2_firstname2.zip

Upload it before the start of the first lab next week. You will get feedback on it.

Make sure to take the quiz. This enables you to check if you understood this first lesson!

5. References

5.1. Typecasting

https://www.learncpp.com/cpp-tutorial/explicit-type-conversion-casting-and-static-cast/

5.2. Function round

http://www.cplusplus.com/reference/cmath/round/?kw=round

5.3. Math functions

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

5.4. Math is fun

https://www.mathsisfun.com

5.5. Initialize random number generator

http://www.learncpp.com/cpp-tutorial/59-random-number-generation/

http://en.cppreference.com/w/cpp/numeric/random/srand

http://www.cplusplus.com/reference/cstdlib/srand/

5.6. Generating random numbers

http://www.cplusplus.com/reference/cstdlib/rand/