VARIABLES PART A

CONTENT

EX02: EXPLORE STRINGS

GENERAL

- Create a new **CONSOLE PROJECT** called **ExploreStrings** in this lab's folder
- Add `#include <string>` to the includes at the top of your cpp file
- Topics: we will explore **`std::string`** and **`char`**, together with **EXPLICIT CONVERSIONS**

ASCII VALUES

Create a function will ask for an ascii value and show the character that goes with it. Choose a clear identifier for it and call it from main.

- ask the user for an ascii value, make sure you store this in an **integer variable**
  - ⇨ next, print both the ascii value and the character that goes with it
  - ⇨ hint: you will need to **convert** your ascii value **to a character**
- example where the user entered *80* :

```
Enter an ASCII value: 80
The ASII value you entered was 80, which results in character P
```

EMAIL GENERATOR (COMBINING STRINGS AND CHARACTERS)

Create a function that will ask for a first and last name, and generate a howest email address based on the information that was entered.

- ask for the first name and save it in a string, do the same for last name
- create a string variable **email** that combines this input to generate a howest email address
  - ⇨ we want to explore adding strings and characters
  - ⇨ therefore, add the dots as a character '.' to the string!
- example result for input "Noah" and "Fence":

```
Enter your first name: Noah
Enter your last name: Fence

Your email:     Noah.Fence@student.howest.be
```

RHYME GENERATOR (CONVERTING NUMBERS TO STRINGS)

Create a function that will ask for a noun, an adjective and a number. Using this input, it generates the first part of a rhyme.

- ask for a noun and an adjective; save each one in a **std::string** variable
- ask for a number as well, save it in an **integer** variable
- create an extra **std::string** variable called rhyme
  - ⇨ use the compound assignment operator **+=** to add the various parts to the rhyme
  - ⇨ structure using <> as placeholders (also see example screenshot further):

    "<number> <adjective> <noun>, lined up in a row.
    One got very tired, so it had to go.
    Now there's only <1 less than number> putting on a show."
  - ⇨ what happens if you do not convert the number to a string before adding it to the rhyme?
  - ⇨ **Hint:** there is a function in the standard (std::) library that can convert an integer to a string. Try to look it up first; ask a teacher if you cannot find it.
- the variable should contain the entire rhyme, print it: `std::cout << rhyme << std::endl;`

- example result for input “squirrels”, “silly” and 10:

```
>> First, give us a noun (plural): squirrels
>> Great! Now pick an adjective: silly
>> Almost there! Enter a number > 2: 10

"10 silly squirrels, lined up in a row.
One got very tired, so it had to go.
Now there's only 9 putting on a show."
```

Any idea how you can print the double quotes “ ” around the rhyme?

COMBINING INTEGERS AND STRINGS

Create a function that asks for an integer and a floating point value. It adds them together as strings and multiplies them as an int and floating point value.

- ask for an integer but save it in a **std::string** variable
- ask for a floating point value as well, but again save it in a **std::string** variable
- add the two strings together and print the result (“1” + “1” = “11” !)
- **convert** the first string value **to an integer**, and the second string value **to a float**
  - ⇨ look up which functions can help you with this; ask a teacher if you cannot find it
  - ⇨ add the two together and show the result (1 + 1 = 2)
- Example result for values “10” and “1.234”:

```
Enter an integer: 10
Enter a floating point value: 1.234

"10" + "1.234" = 101.234
10 + 1.234 = 11.234
```

EX03: EXPLORE MATH

-- We will explore existing math functions, first in console, next we use them to draw. All functions needed with info and example code can be found in the cmath reference –

GENERAL

- Create a new **PROG1 GAME PROJECT** called **ExploreMath** in this lab’s folder
  - do not forget to set the build dependency
- Topics: we will explore some `cmath` functions
- For example, to round up a floating point value:

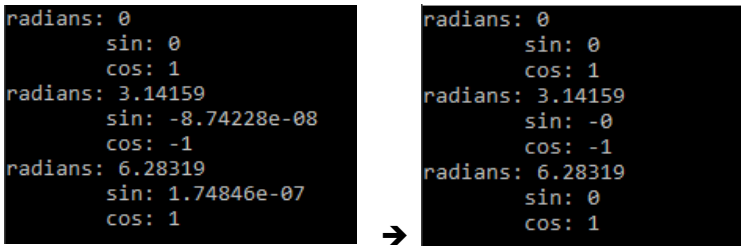
```
float value{ 1.234f };
int number{ int( ceil( value ) ) };   //reference: ceil, result: 2
```

PRINT A TEST TABLE

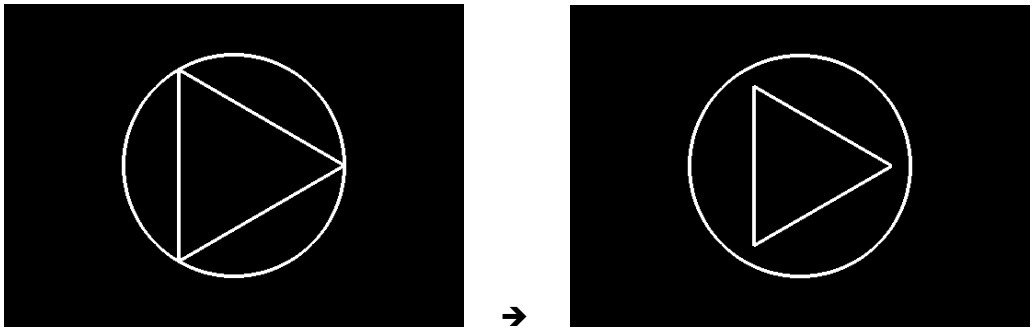
Create a function that will print a table with the sin and cos values of some angles (see further).

- **WARNING:**
  - ➢ To **declare** functions in a game project, add them to the **game.h** file! (Find the region “`ownDeclarations`”).
  - ➢ Next, implement your functions in the **game.cpp** file (find the region " `ownDefinitions`” at the bottom).
- in the function, create a local float variable pi which contains a value for pi, eg. 3.1415926535

- create a second float variable called angle, which will contain an angle in **radians**
  - **Hint:** all functions you need can be found in the cmath reference
  1. initialize the angle with **0**, and print its cos and sin value
  2. change the angle's value to **pi** and print again
  3. change it to a full circle (**2 times pi**) and print again
  - Call the function from `Draw()` first; the result appears in the console window behind the game window. Next, only call it from `Start()` instead. What difference do you notice?
  - check the screenshots below: the left shows the scientific notations of some decimals; fix this by rounding the values to get the result on the right:



HANDS-ON DEMO: USING TRIGENOMETRY TO DRAW



- create a function to draw a "play button" and call it from draw
- draw a circle with a thick stroke, make sure to use variables for both the position and the radius
- use those variables in combination with trigonometry to draw the 3 lines as above in the left screenshot; which formulas do you need?
- next, try to apply the necessary changes to go from the result in the left screenshot to that in the right

EX04: STATIC DRAWING

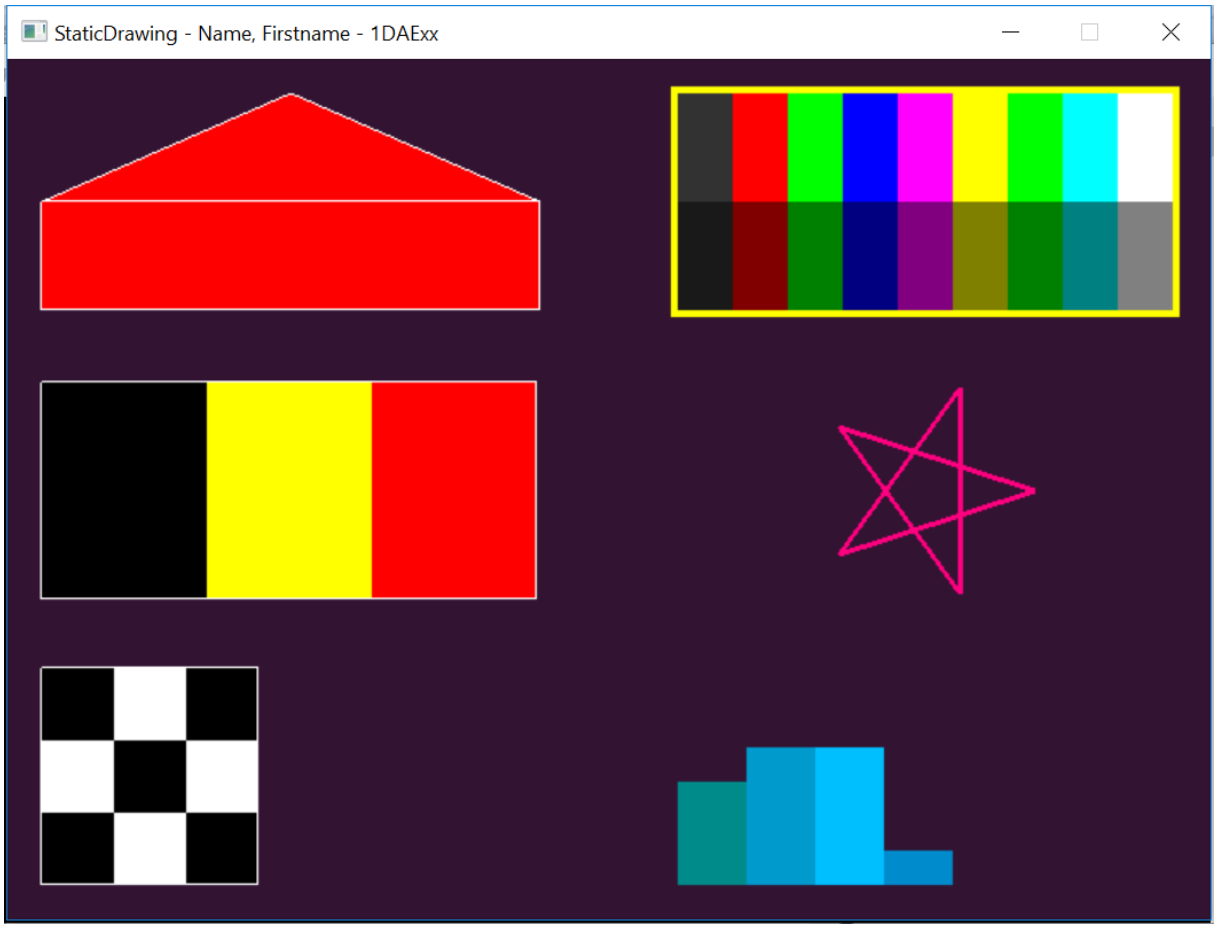
*If you already have programming experience, you can skip the exercises that might seem repetitive and spend more time on the more challenging and the free exercise described at the end of this document. However, do the **pentagram** and **columns graph** exercises as you'll animate them in one of the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

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

GENERAL

Create a new **Prog1 Game Project** called **StaticDrawing** in your 1DAEXX_02_NAME_FIRSTNAME folder. 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.



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

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.

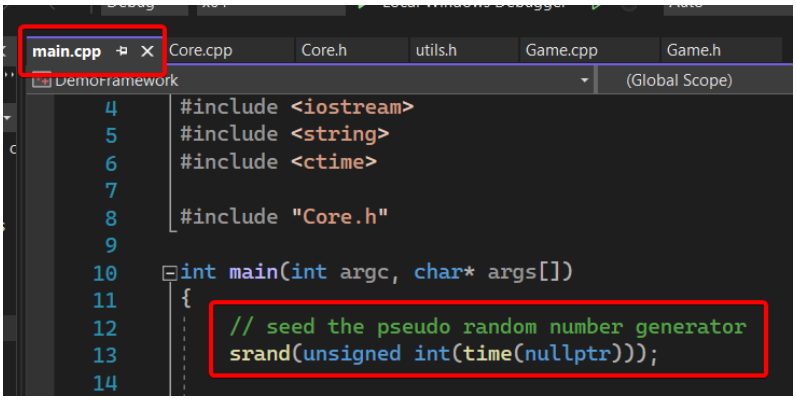
EX05: RANDOM RECTANGLES

Create a new Prog1 Game Project with name **RandomRectangles** in your 1DAEXX_02_NAME_FIRSTNAME folder. Adapt the window title

WARNING 1: If you are sensitive to high frequency screen flashes, DO NOT MAKE THIS EXERCISE.

WARNING 2: For **game project (graphical) applications** the seeding srand already happens in the main function

- **do not call it again in draw as it will slow down the draw loop significantly!**



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.

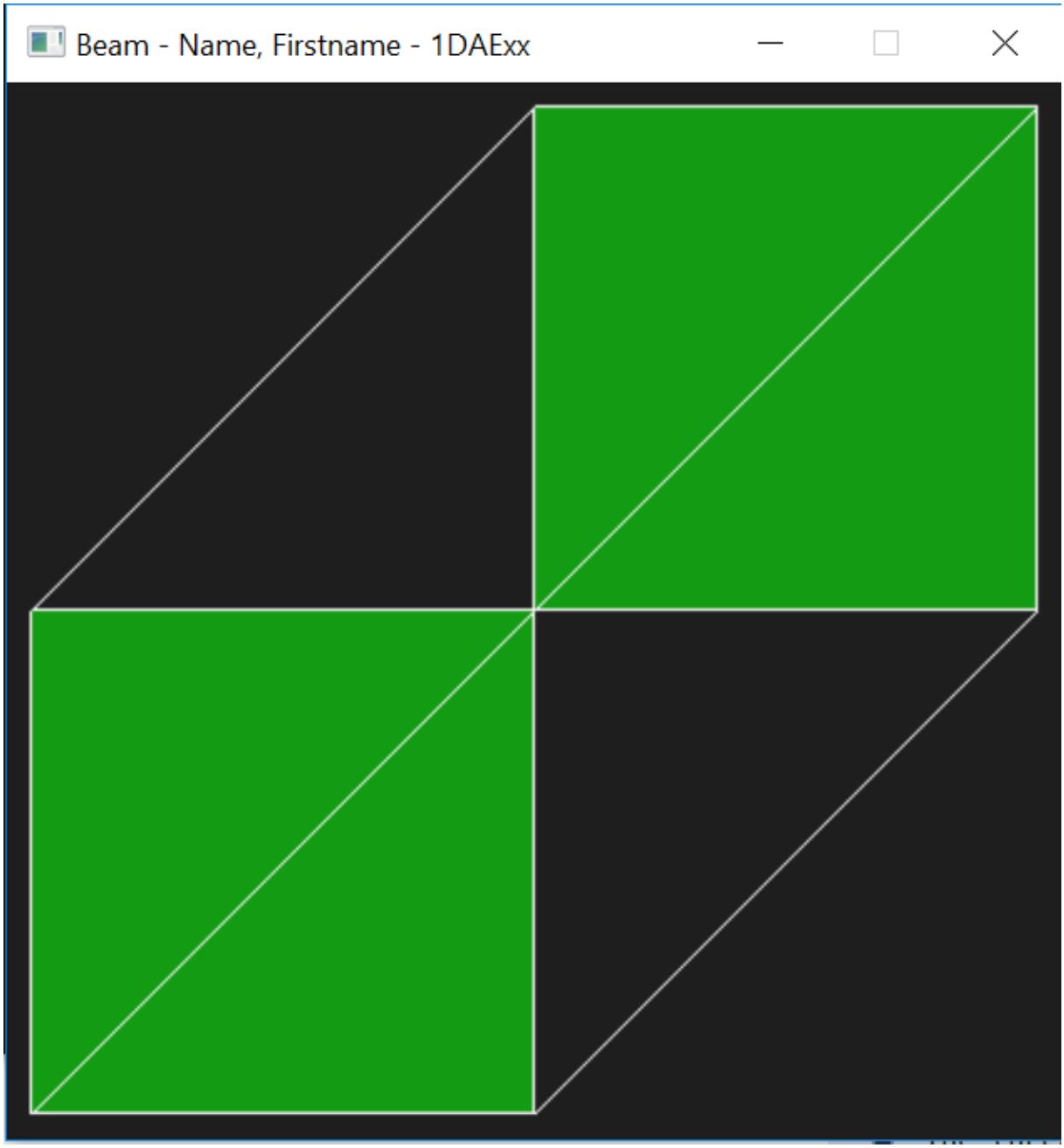
EX06: BEAM

Create a new game project with name **Beam** in your 1DAEXX_02_NAME_FIRSTNAME folder.

- Adapt the window title.

EXERCISE:

- Write the code to get the image below
  - take into account the restrictions as listed in next paragraph.
- *(see next page)*



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 (or magic numbers) are NOT allowed.
    - **Note:** by using g_WindowWidth and g_WindowHeight, we can always resize or window and everything should just move along with it.

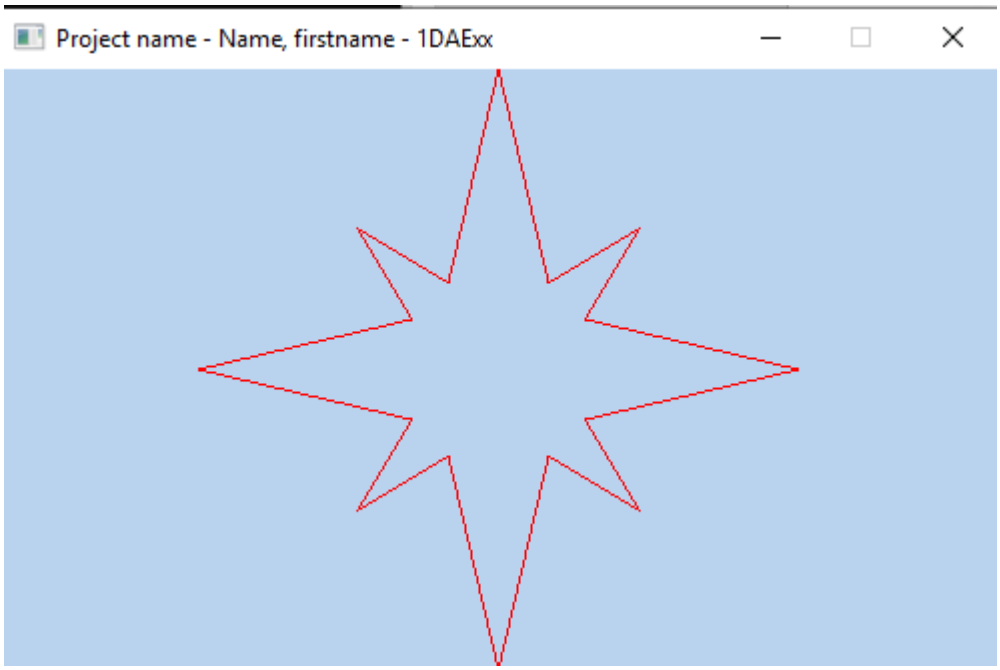
CHANGE THE WINDOW SIZE

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

EX07: STAR

Create a new game project with name **Star** in your 1DAEXX_02_NAME_FIRSTNAME folder and set the title.

- Change the title of the window and adapt the window size to 500 x 500.
- Then make the exercise as described further. In the end, you should have a window that looks like this *(see next page); remember the **hands-on demo** we provided earlier.*
- Below is a more detailed description on how to proceed.



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**
  - indicate on it the names you give to the 32 vertices coordinates.
  - Use it to think in advance what you need for your formula, for example which angles, radiuses, etc.
- Follow the steps / hints below if necessary to get to the result.

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.

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

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.

DRAW THE STAR

- Draw lines connecting the points.

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.

EX06: CODING CRAFTSMANSHIP CHALLENGE

Level up your game development powers! This is where creativity meets code, and it bridges the gap between the traditional step-by-step written lab assignments and a much deeper understanding of the material.

In this weekly exercise you're tasked with inventing an exercise that merges all c++ programming and game development skills you obtained so far in this course into a (small!) unique project. Let your imagination run wild and work on a programming project YOU love!

Do not underestimate the value of this exercise; experimenting is by far the best way to master programming!

Create either a **console** **or** a **Prog1 Game Project** called CodingCraftmanship02 in the 1DAEXX_01_NAME_FIRSTNAME folder.

Use as much of the new topics as possible: functions, math functions, conversions, std::string, … to create a small game for example.

SUBMISSION INSTRUCTIONS

You have to upload the folder *1DAExx_02_name_firstname.*

- ***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 or rar file.

- 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!

REFERENCES

EX08: STRUCTURE OF A PROGRAM

http://www.cplusplus.com/doc/tutorial/program_structure/

EX09: COUT, CIN, ENDL, STD, USING

A first look at cout, cin, endl, the std namespace, and using statements.

http://www.learncpp.com/cpp-tutorial/1-3a-a-first-look-at-cout-cin-endl-namespaces-and-using-statements/

EX10: LITERALS

http://www.learncpp.com/cpp-tutorial/28-literals/

EX11: OPERATOR PRECEDENCE AND ASSOCIATIVITY

http://www.learncpp.com/cpp-tutorial/31-precedence-and-associativity/

EX12: COMPILING AND LINKING

http://www.cprogramming.com/compilingandlinking.html