# Variables part 3

## 1. Content



## 2. Objective

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

- Know and work with unsigned variable types
- Getting the size of variables
- Use the bitwise operators
- Explain the range of integer type variables and what overflow means
- Use const keyword in variable definitions
- Know what the difference is between local and global variables.
- Create animated graphics

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. **Variables2Basics**. Other names will be rejected.

## 3.1. Const

Reference: C++ Coding Standards. Item 15:
"Immutable values are easier to understand, track, and reason about, so prefer constants over variables wherever it is sensible and make **const** your default choice when you define a value: It's safe, it's checked at compile time (see Item 14), and it's integrated with C++'s type system."

Variables that will never change should be const. e.g.

```
const double pi{3.1415926535};
```

**So, from now on always apply this rule.**

## 3.2. Variables3Basics

Create a new **project** with name **Variables3Basics** in your **1DAExx_03_name_firstname** folder.

In this application, you'll make some more basic exercises on variables and operators and you'll generate random numbers.

### 3.2.1. Sizeof operator

The sizeof operator is used to know the size in bytes of an object or variable.

Create and initialize variables of bool, int, float and double. Use the sizeof operator to get their size and print the result on the console.

### 3.2.2. Hexadecimal and binary number presentation

Assign 3 times the same value (e.g. decimal 12) to an integer type variable using literals of different numeral system: **decimal, hexadecimal and binary literal** values, and each time print the value to the console. Look up on this link here how to indicate that a literal is hexadecimal or binary.

Following table shows the first 17 positive integers in these numeral systems

| Decimal (0-9) | Binary (0-1) | Hexadecimal (0-F) |
|---|---|---|
| 0 | 0000 0000 | 0 |
| 1 | 0000 0001 | 1 |
| 2 | 0000 0010 | 2 |
| 3 | 0000 0011 | 3 |
| 4 | 0000 0100 | 4 |
| 5 | 0000 0101 | 5 |

| | | |
|---|---|---|
| 6 | 0000 0110 | 6 |
| 7 | 0000 0111 | 7 |
| 8 | 0000 1000 | 8 |
| 9 | 0000 1001 | 9 |
| 10 | 0000 1010 | A |
| 11 | 0000 1011 | B |
| 12 | 0000 1100 | C |
| 13 | 0000 1101 | D |
| 14 | 0000 1110 | E |
| 15 | 0000 1111 | F |
| 16 | 0001 0000 | 10 |
| 17 | 0001 0001 | 11 |

### 3.2.3. Range of – signed vs unsigned types

The table below shows the range of values a type can store:

| Type | #bytes | Min | Max |
|---|---|---|---|
| char | 1 | -128 | 127 |
| unsigned char | 1 | 0 | 255 |
| int | 4 | -2147483648<br>Or<br>`INT32_MIN` | 2147483647<br>Or<br>`INT32_MAX` |
| Unsigned int | 4 | 0 | 4294967295<br>or<br>`UINT32_MAX` |

What happens if you add/subtract a value so that an overflow occurs? To find out, define 4 **integer** variables: 2 of type **unsigned** and 2 of type **signed**.

Give them a value equal the min/max boundaries. Then subtract/add 1 to the variable and see what happens.

Read about this on Integer overflow

### 3.2.4. Binary bitwise | and & operator

First read about these operators on Bitwise operators

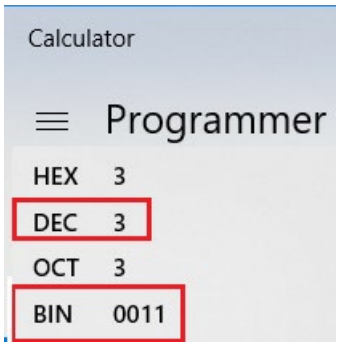
#### a. Bitwise & (and) operator

First, make the exercise on paper.

- Consider 2 integers with values 3 and 5
- Write these values down in binary under each other, putting corresponding bits in the same column

- Then apply the & operation on the bits in the same column
- Convert the resulting bits to a decimal

The windows calculator also has this functionality in the Programmer view.




Then verify whether you get the same result by writing the code

- Define 2 unsigned int variables with values 3 and 5
- Use the &-operator with these variables as operands
- Send the result of the operation to the console

### b. Bitwise | (or) operator

Make the same exercise on paper however use the | operator

Then check the result by writing and executing some code.

### c. Applied: Bitwise &: is a bit 0 or 1?

The bitwise & is often used to verify whether a bit (flag) is set in a number.

Write some code that tells whether the 1st, 2nd, 3rd, or 4th bit (this counting is started at the least significant bit) is set in a number entered by the user. If the result is 0 then that bit is not set, just show the result.

```
Number to check which bits are set? 12
1st bit 0
2nd bit 0
3rd bit 4
4th bit 8
```

### d. Applied: Bitwise |: set a bit in a number

The bitwise or is often used to set a bit (flag) in a number.

Ask the user to enter an integer number and then set the 3rd flag in this number and print the result.

```
Number to set 3rd bit in? 8
This number with 3rd bit set 12
```

**e. Bitwise exclusive or (XOR)**

```
01110010 ^
10101010
--------
11011000
```

if both corresponding bit inputs are 1 or both inputs are 0, it returns 0. Else it returns 1. No programming is needed for this one.

**f. Right shift (divide unsigned integer by 2)**

Used on unsigned types only ( sign bit !)

Moves all the bits to the right, zero's are added to the left.

Define a variable with value 2048, print and demonstrate the right shift operator.

Check whether the bit representing 256 is set in the value 4448. **The printed result must be 0 or 1 without using conditional expressions** use bit shifting. This is not an academic exercise; this is often used to produce very fast code.

**g. Left shift (multiply unsigned integer by 2)**

Used on unsigned types only ( sign bit !)

Moves all the bits to the left, zero's are added to the right.

Define a variable with value 2048, print and demonstrate the right shift operator.

**h. General bitwise operator exercise.**

Define a variable with the value 0. Find a way to turn this into 2845631 only using the bitwise operators we discussed in this chapter. To be more precise: know what bits in that number need to be '1'. Then shift 1 to the left for each bit that needs to be set and or the result. E.g. 12 is (1 << 3) | ( 1 << 2).

## 3.3. ErrorSolving

Create a new project with name **ErrorSolving** in your **1DAExx_03_name_firstname** folder.

Overwrite the generated **ErrorSolving.cpp** file by the given one.

First solve all build warnings and errors. Hereby proceed as follows:

1. Solve the first error/warning
2. Then build again.
3. If the build reports errors/warning, then start again at 1

When all build errors are solved, solve the runtime errors.

**If you want to become a good programmer, you should be able to explain the cause of these problems/errors, solving them is not enough.**

## 3.4. Animated Drawings

### 3.4.1. Adding functions to the framework

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.

### 3.4.2. Local vs global variables

Read this about local vs global variables:

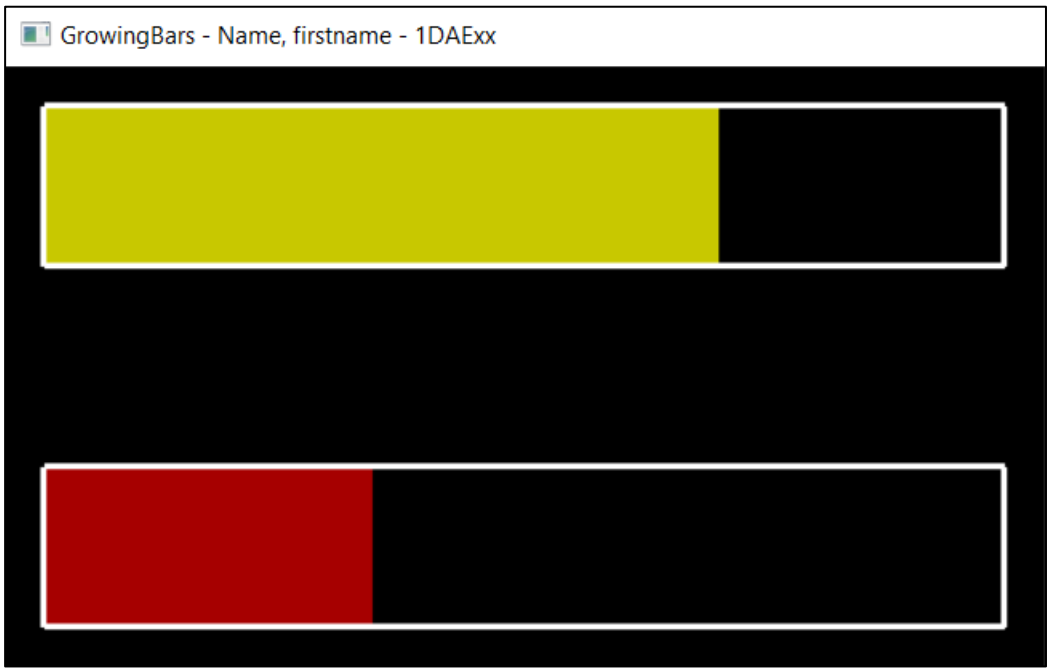
https://www.tutorialspoint.com/What-are-local-variables-and-global-variables-in-Cplusplus

### 3.4.3. GrowingBars

Create a new framework project with name **GrowingBars** in your **1DAExx_03_name_firstname** folder.

Delete the generated **GrowingBars.cpp** file.

Follow the steps as described further.




Adapt the window title.

Then add code that counts the frames in a global variable **g_NrFrames** and draws 2 animated bars using the value in this variable:

- A first yellowish **bar** that grows one pixel every frame, its color does not change and it restarts at a width of zero when the maximum width is reached (use the modulo operator). A white border indicates the maximum size of the bar.
- A second reddish **bar** that grows 15 pixels every 30 frames. When the maximum width is reached, it also restarts at a width equal to 0. A white border indicates the maximum size of the bar.
  Optional: Make the bar change color. The bar starts with a red color equals

120 and while growing a same portion red is added so that it gets the max value 255 when it reaches its max width.

Do not forget to keep your code easy-to-read, e.g. define two Draw… functions that draw those bars.

### 3.4.4. MovingLines

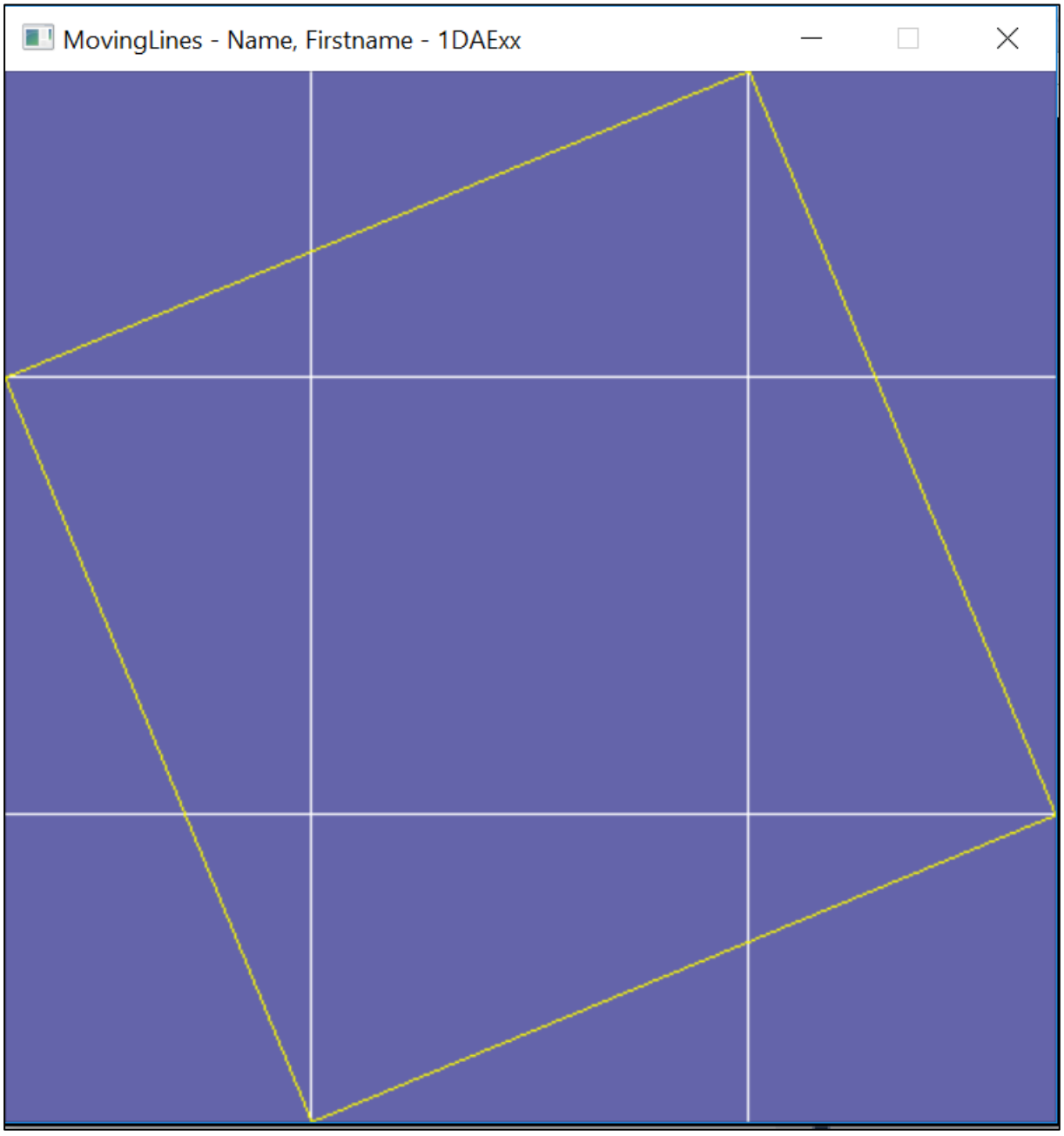
Create a new framework project with name **MovingLines** in your **1DAExx_03_name_firstname** folder.

Delete the generated **MovingLines.cpp** file.

Change the size of the window into 500 x 500.

Adapt the title.

Make the exercises as described further in this document. At the end, you should have a window that looks like this. You find more information below the screenshot.



Draw a blue background and the following 4 moving white lines:

- A 1st horizontal line that starts at the top of the window and moves downwards. The line moves 1 pixel each frame.
- A 2nd horizontal line that starts at the bottom and moves upwards (it is possible to do this without adding extra variables).
- A 1st vertical line that starts left and moves to the right (without adding extra variables).
- A 2nd vertical line that start at the right and moves to the left (without adding extra variables).

Use the modulo-operator to make the animation infinite: example: when the second horizontal line reaches the top (height of window), make its vertical position get the value 0 again using the modulo-operator.

Add another 4 yellow lines that connect the previous lines as shown in the screenshot.

### 3.4.5. SlidingRectangles

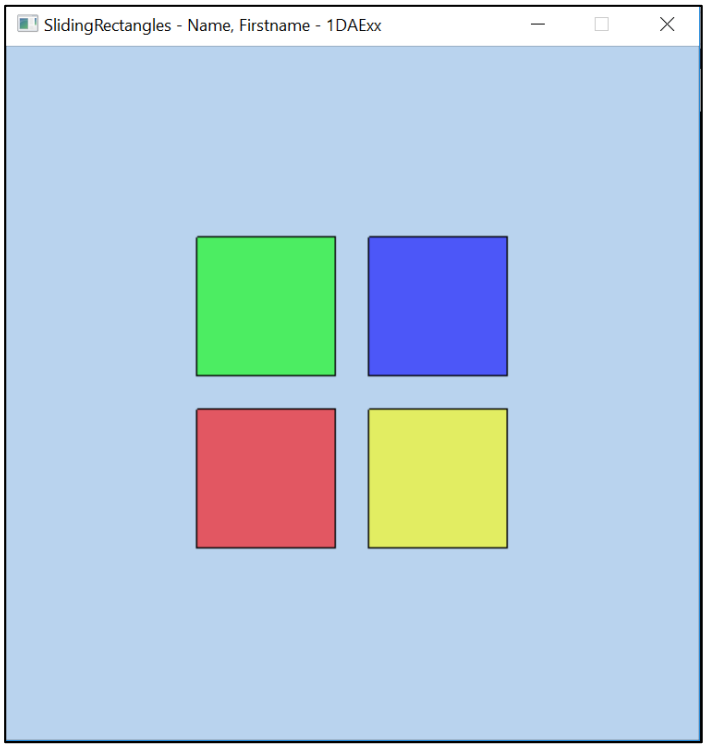
Create a new framework project with name **SlidingRectangles** in your **1DAExx_03_name_firstname** folder.

Delete the generated **SlidingRectangles.cpp** file.

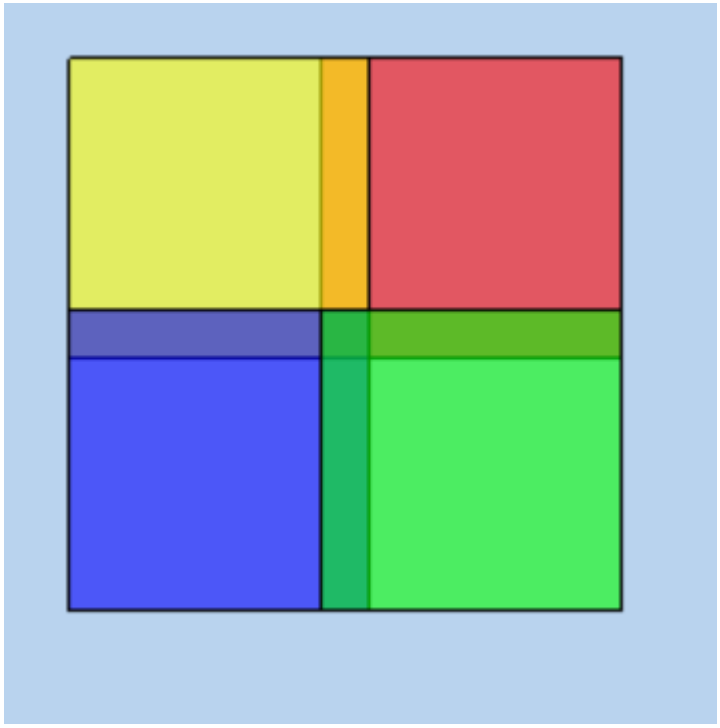
The window should have the same width and height.

Adapt the title.

In this project you'll draw 4 semi-transparent squares that slide between opposite window corners at a speed of 1 pixel per frame in both horizontal and vertical direction. At the end of this exercise you should have a screen that looks like this.




When they are overlapping



### 3.4.6. ClockPointers

This is an application that combines math skill with programming skills. That is very powerful and awesome! That is what raytracing and rasterization is all about. You will need to use trigonometry (sin, cos) to rotate these points about the center of the window. Don't panic if you can't finish this on. It's one of the last applications and these are the hardest to implement.

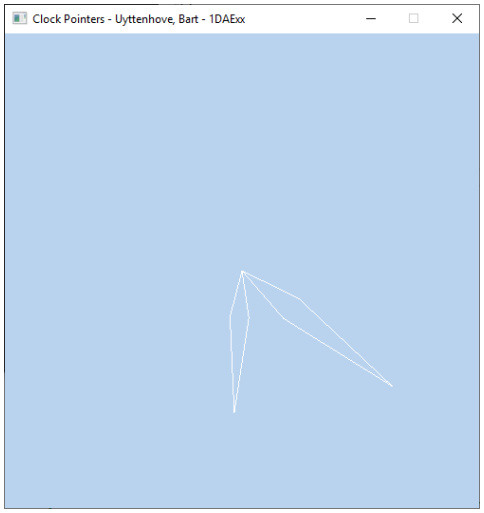
Create a new framework project with name **ClockPointers** in your **1DAExx_03_name_firstname** folder.

Delete the generated **ClockPointers.cpp** file.

Change the window title.

In this project you'll draw 2 rotating clock pointers with different length and different angular velocity.

- The larger one makes a complete rotation in 240 frames.
- The smaller one rotates 12 times slower than the larger one.



At the start both pointers make an angle of 0 degrees with the positive x-axis, so they are parallel to the x-axis and point to the right.

Always work step by step towards the solution, for instance:

1. Have angle (units are radians) variables that are incremented each frame.
2. Take the cos/sin of the angles, this results in values between -1 and +1.
3. Scale up these values to e.g. -100 to +100.
4. Translate all the values to the center of the window. (add width/2, etc…)
5. Draw a line from the center of the window to the resulting values.

Now you should have two rotating lines.

6. For each line, one coordinate was calculated, now add two coordinates: one with a greater angle, and one with a smaller angle. They are 1/3 of the line lengths away from the center.
7. Replace each line with 4 lines connecting the coordinates.

### 3.5. FreeAnimation

**Practice makes perfect.**

Create a new framework project with name **FreeAnimation** in your **1DAExx_03_name_firstname** folder.

Delete the generated **FreeAnimation.cpp** file

Add it to your Visual Studio project and change the window title.

Think yourself about an exercise that shows some animated drawings.

## 4. Submission instructions

You have to upload the folder *1DAExx_03_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** or delete the debug folder.
- Then **close** the project in Visual Studio.
- Delete the .vs folder.

Make sure that you answered the quiz.

Compress this *1DAExx_03_name_firstname* folder and upload it before the start of the first lab next week.

## 5. References

### 5.1. Binary literals

https://en.cppreference.com/w/cpp/language/integer_literal

### 5.2. Bitwise operators

http://www.learncpp.com/cpp-tutorial/38-bitwise-operators/

https://www.bogotobogo.com/cplusplus/quiz_bit_manipulation.php

## 5.3. Integer overflow

http://www.cplusplus.com/articles/DE18T05o/

https://www.ima.umn.edu/~arnold/disasters/ariane.html