Flow control

1. Content

2. Objective

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

- Use the logical &&, || and ! operators
- Use the selection statements:
  - switch
  - if - else if – else
- Define and use enumerations
- Define and use new types using struct
- Use IO manipulators to change the precision and width of output
- Handling input events from the user:
  - keypress,
  - moving/clicking the mouse,
  - close the application.

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.

While making the exercises, answer the questions listed in the Q&A document located in the 1DAExx_04_name_firstname folder.

Make the labs in the same order as this assignment

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

3.1. FlowControlBasics

Create a new **project** with name **FlowControlBasics** in your **1DAExx_04_name_firstname** folder.

Make following exercises. Keep in mind that your code should be easy-to-read, e.g. create a function for each exercise. We defined following functions, notice we use upper camel casing for the function name, this is part of our naming standards:

```
void PrintTruthTable();
void CompareTwoNumbers();
void CalculatePrice();
void ConvertDayNumber();
void CheckLeapYear();
void ConvertSeconds();
void UseActorState();
void UsePoint2f();
```

Another advantage of using functions in this project is that you do not need to execute all the functions every time you launch the application, just comment the ones that run fine.

```
//PrintTruthTable();
//CompareTwoNumbers();
//CalculatePrice();
//ConvertDayNumber();
//CheckLeapYear();
ConvertSeconds();
//UseActorState();
//UsePoint2f();
```

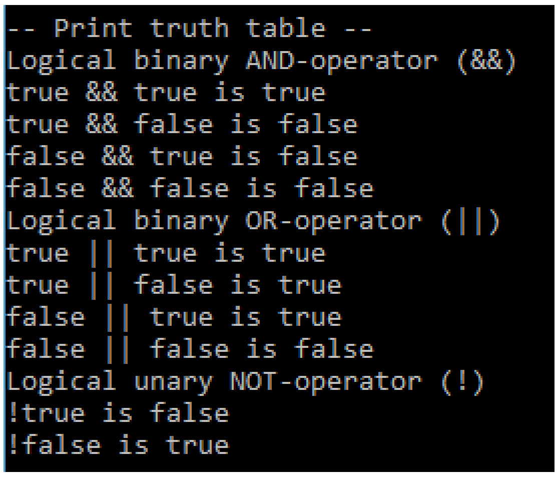
3.1.1. Logical operators

a. Print the truth table

- Define 2 bool variables, one having the value *true* the other the value *false*.
- When sending a bool variable to the output stream a 1 or 0 is printed. But if you first add std::boolalpha to the output stream, then "true" or "false" will be printed instead.

```
std::cout << std::boolalpha;
```

- Use this link to know how the Operator Precedence C++ is organized.
- Make all possible combinations of both bool variables using the logical AND-operator (&&) and show the results on the console.
- Make all possible combinations of both bool variables using the logical OR-operator (||) and show the results on the console.
- Apply the logical NOT-operator (!) on the bool values and show the result on the console.

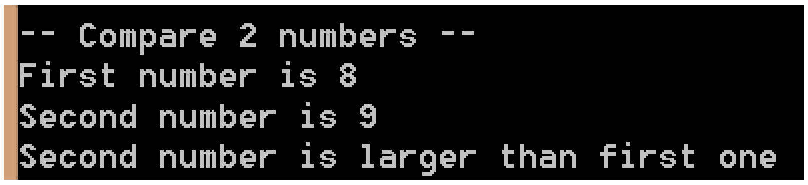


```
-- Print truth table --
Logical binary AND-operator (&&)
true && true is true
true && false is false
false && true is false
false && false is false
Logical binary OR-operator (||)
true || true is true
true || false is true
false || true is true
false || false is false
Logical unary NOT-operator (!)
!true is false
!false is true
```

3.1.2. Switch and if statements

a. Compare two numbers

Generate two random numbers in the inclusive interval [0, 10]. Print both numbers followed by a message that indicates which one is larger. Ensure that you don’t always have the same sequence of random numbers.



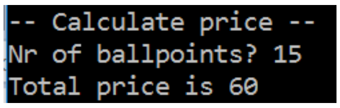
```
-- Compare 2 numbers --
First number is 8
Second number is 9
Second number is larger than first one
```

b. Calculate price

A company sells ballpoints in large quantities. Following prices apply:

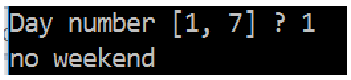
Number	Interval	Price a piece
Less than 20	[0,19]	4,00 €
20 to 49	[20,49]	3,50 €
50 to and including 100	[50,100]	3,25 €
Over 100	[101,…	3,10 €

Let the program ask the number of ballpoints and show what the total price is.

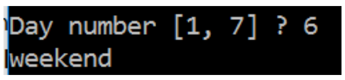


```
-- Calculate price --
Nr of ballpoints? 15
Total price is 60
```

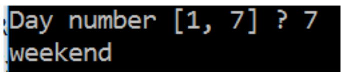
c. Convert day number



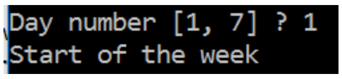
```
Day number [1, 7] ? 1
no weekend
```



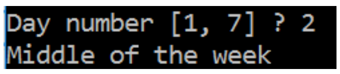
```
Day number [1, 7] ? 6
weekend
```



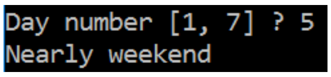
```
Day number [1, 7] ? 7
weekend
```



```
Day number [1, 7] ? 1
Start of the week
```

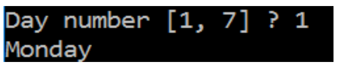


```
Day number [1, 7] ? 2
Middle of the week
```

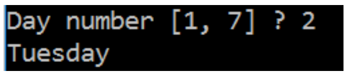


```
Day number [1, 7] ? 5
Nearly weekend
```

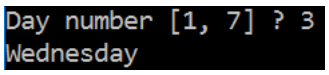
Day	Message
1	Start of the week
[2,4]	Middle of the week
5	Nearly weekend
[6,7]	Weekend
Other	Wrong day number



```
Day number [1, 7] ? 1
Monday
```

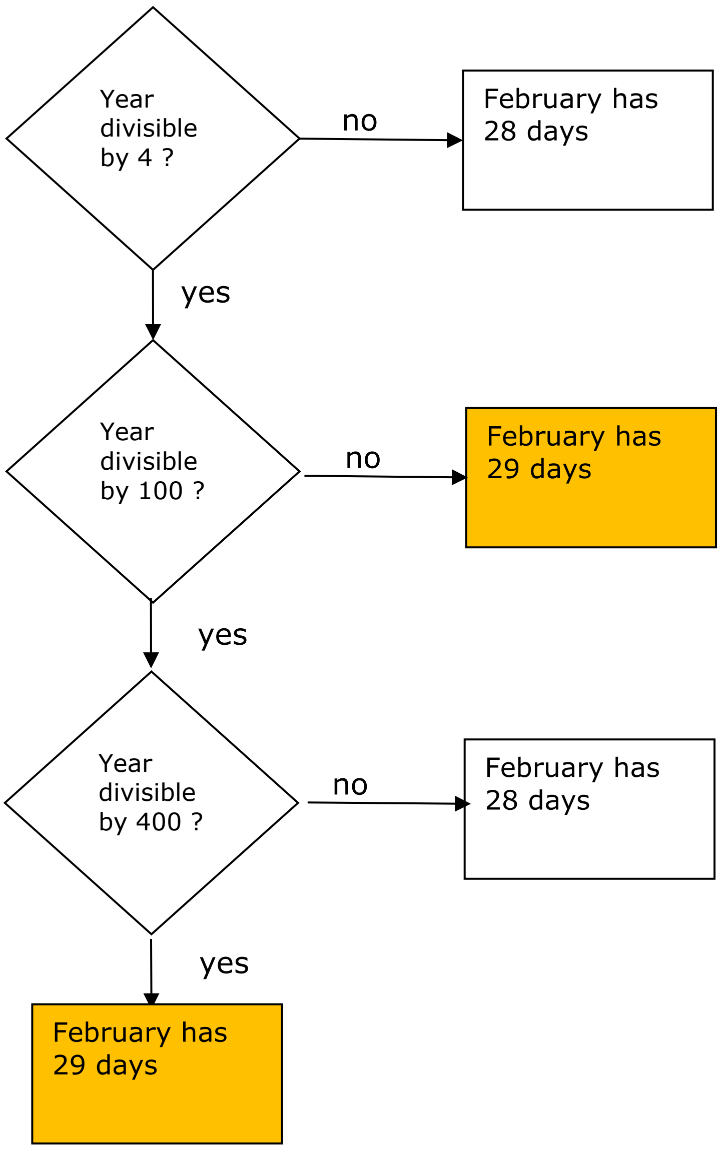


```
Day number [1, 7] ? 2
Tuesday
```



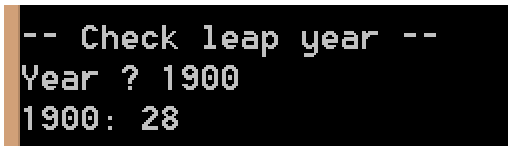
```
Day number [1, 7] ? 3
Wednesday
```

d. Check for leap year



A **leap year** is a year containing one additional day in February (29 instead of 28 days). The flow chart above shows how you can determine whether a year is a common or a leap year.

- Ask the user to enter a year.
- Print a message telling the number of days in February in that year.



```
-- Check leap year --
Year ? 1900
1900: 28
```

Always test your code properly. You should test each possible branch in the selection statement. Using next values all branches are tested.

Year	Number of days in February
2014	28
2012	29
2000	29
1900	28

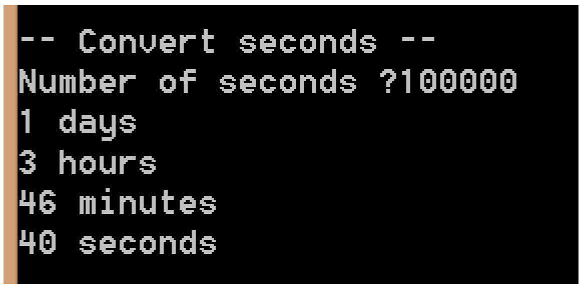
e. Convert seconds

The user enters a number of seconds.

Calculate the number of days, hours, minutes and remaining seconds that the entered number of seconds contains.

Show the results in the console.

Tip: Use the modulo-operator.



```
-- Convert seconds --
Number of seconds ?100000
1 days
3 hours
46 minutes
40 seconds
```

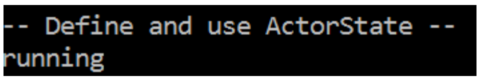
3.1.3. Scoped enumerations

For information on enumerations see the theory and Enumeration

Define a scoped enumeration class **ActorState** with 3 possible values (enumerators): *running*, *flying* and *falling*.

Define a variable of this new type and initialize it with one of the 3 enumerators.

Test this variable using a switch case and print an appropriate message for each of the possible enumerator values, e.g. when initialized with the *running* enumerator.

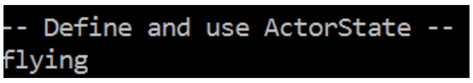


```
-- Define and use ActorState --
running
```

Change the code to use a random value for this state variable. It is not possible to randomize an enumeration directly. But you can proceed as follows:

- Generate a random number within the right interval (you can cast the first and last enumerator to an int to know the interval boundaries).
- Cast this random integer number to ActorState type and use it to initialize the ActorState variable

Again use a switch statement to test the content of the variable and show an appropriate message on the console.



```
-- Define and use ActorState --
flying
```

3.1.4. Using the struct Point2f

There are many cases in programming where we need more than one variable in order to represent an object. Using the **struct** keyword you can define new types that contain one or more other types. For more information see the video and the url here: Structs.

A **Point2f** struct is defined in structs.h, have a look at it.

Extract structs.cpp and structs.h from the framework files and add them to your project.

The struct members can be accessed through the . operator. E.g. myPoint.x

Define 3 variables of this new type in different ways:

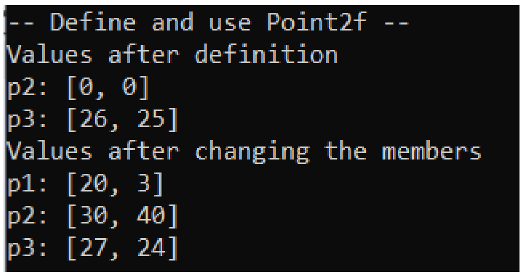
- p1: without initialization.
- p2: with uniform default initialization.
- p3: with uniform initialization using literals e.g. 26 and 25

Print the Point2f values on the console, you can access their elements x and y using the dot operator. This leads to a problem for p1, because it was not initialized. Don’t print that one, comment that code. Remember that you should always initialize local variables.

Then change their values like this:

- p1: assigning the value 20 to x and the value 3 to y
- p2: assigning the value Point2f {30,40}
- p3: applying the ++ and -- operator on x and y

And print them again, now p1 is printed without any problems because a value was assigned to it.



```
-- Define and use Point2f --
Values after definition
p2: [0, 0]
p3: [26, 25]
Values after changing the members
p1: [20, 3]
p2: [30, 40]
p3: [27, 24]
```

3.2. SDLEvents

This project will be demonstrated by the teacher (or in a video).

Now that you know iterations and selection statements, time to add event handling to your projects.

Create a new sld project with name **SDLEvents** in your **1DAExx_04_name_firstname** folder.

Delete the generated **SDLEvents.cpp** file.

Update the window title.

Until now you could only get user input from the console. In this project we will show you how you can get input related to the SDL window.

For more information see Event driven programming

3.2.1. Run function

In the main.cpp , you see that in the main function, the Run function is called. The definition of the Run function can be found in the core.cpp file

Open the core.cpp file.

The Run function contains a loop consisting of:

1. A call to the Start function that is used to load game resources such as images and sounds (later).
2. **Code that handles input (events) from the user**, such as:
   - pressing a key while the window has the focus,
   - moving/clicking the mouse on the window,
   - closing the window…
   - these functions are in the main.cpp file.
3. Code that calculates the time between two draw calls
4. A call to the update function that is in the main,cpp.
5. A call to the Draw function (main.cpp)
6. Swapping front and back buffer (SDL_GL_SwapWindow)

The code that handles user input requires some extra explanation.

3.2.2. Polling events

SDL stores events in an event queue. With the **SDL_PollEvent** function, you get information about the next event in this event queue. And it also removes that event from the queue. This function results in an int type, that has the value:

- 1 when there was an event available in the queue,
- 0 when none was available.

Information about an event is wrapped in an **SDL_Event** type variable that is passed as an argument to this function. The function puts information about the polled event in this argument.

When there are no longer events in the queue, then we execute the draw operation.

For more information see SDL event

3.2.3. Handle the events

a. Trace some common events

Let's handle some events using a switch statement. Show an appropriate message on the console for every event function in Game.cpp.

b. Get extra event information

SDL_Event contains next to the event type several pieces of extra information.

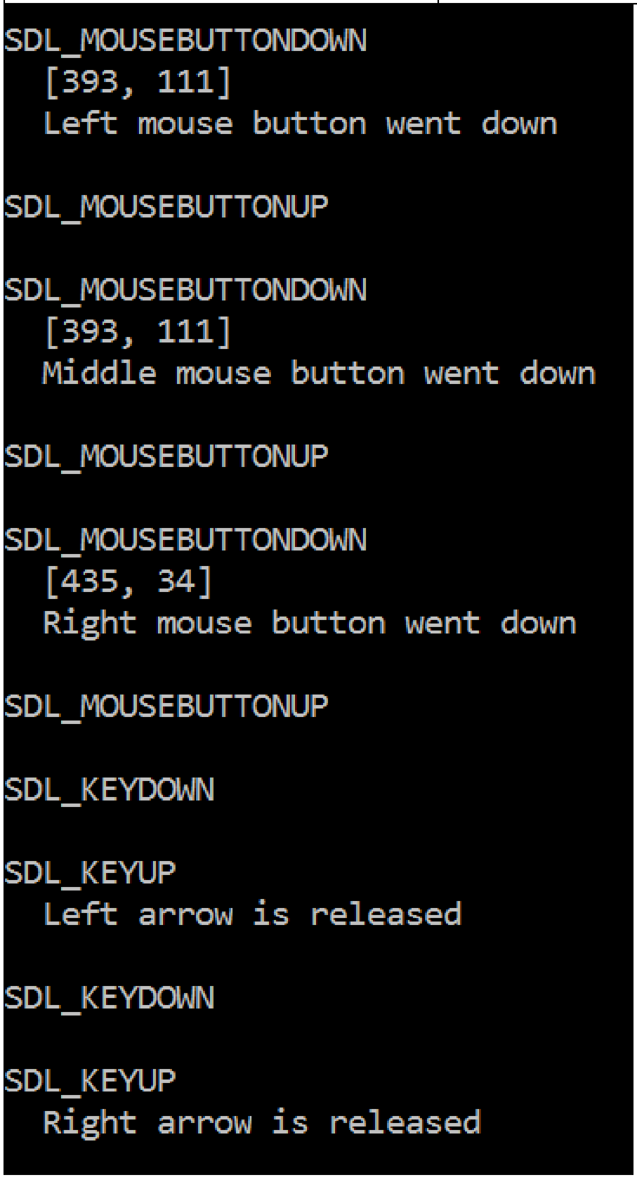
Show extra information on the console as indicated in next table and screenshot.

In particular show a message when the left/right arrow key is released and when the left, middle and right mouse button went down.

In the functions, there is commented code that can be quite useful. To find out how other keys can be checked, click SDLK_LEFT and press F12. A new window opens showing a sdl header file.

Event type (e.type)	Extra information	In which event part

SDL_KEYUP SDL_KEYDOWN Keyboard button event SDL Keycode Lookup Table	Which virtual key	e.key.keysym.sym
SDL_MOUSEMOTION Mouse motion event	Mouse position	e.motion.x , motion.y
SDL_MOUSEBUTTONDOWN SDL_MOUSEBUTTONUP Mouse button event	Mouse position	e.button.x , button.y
	Which button	e.button.button



```
SDL_MOUSEBUTTONDOWN
  [393, 111]
  Left mouse button went down

SDL_MOUSEBUTTONUP

SDL_MOUSEBUTTONDOWN
  [393, 111]
  Middle mouse button went down

SDL_MOUSEBUTTONUP

SDL_MOUSEBUTTONDOWN
  [435, 34]
  Right mouse button went down

SDL_MOUSEBUTTONUP

SDL_KEYDOWN

SDL_KEYUP
  Left arrow is released

SDL_KEYDOWN

SDL_KEYUP
  Right arrow is released
```

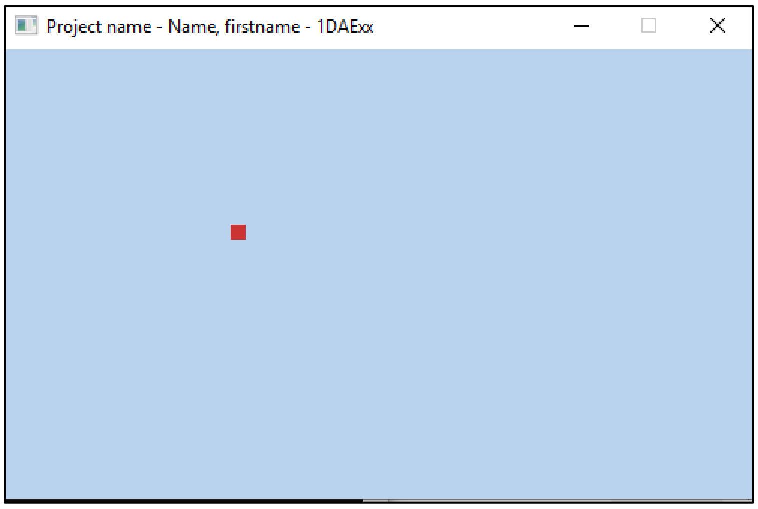
c. Fill a little square on the mouse position

Let's visualize the position of the mouse. Therefore, you'll need to keep track of the mouse position in a global variable as you need this variable in 2 methods: OnMouseMotionEvent and Draw.

Define a Point2f type variable to hold the x and y coordinates of the mouse position.

In the OnMouseMotionEvent function, change the values of the Point2f variable when a **mouse move event** occurs.

In the Draw function, fill a little square with size 10 on this position.



3.3. ErrorSolving

Create a new project with name **ErrorSolving** in your **1DAExx_04_name_firstname** folder.

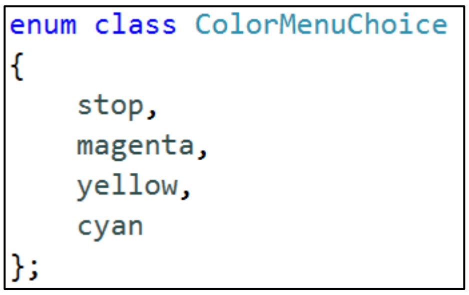
Overwrite the generated file **ErrorSolving.cpp** by the given one.

3.3.1. Runtime errors

This code builds without problems, but the application doesn't behave as it should.

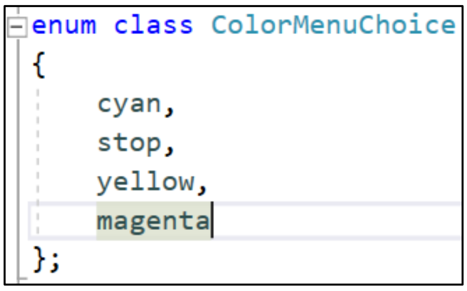
3.3.2. Use a scoped enumeration

When all these problems are solved, then define and use a scoped enumeration instead of magic numbers in the **GetColorCodes** function. Remember that enum types can be typecasted to int types. The menu string and the switch structure should use this enumeration.



```
enum class ColorMenuChoice
{
    stop,
    magenta,
    yellow,
    cyan
};
```

Test all enumerators. Then only change the sequence of the enumerators, everything should still work correctly.



```
enum class ColorMenuChoice
{
    cyan,
    stop,
    yellow,
    magenta
};
```

3.4. Events and Drawing

Adding your own functions to a framework project.

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.

Create a new SDL project with name **EventsAndDrawing** in your **1DAExx_04_name_firstname** folder.

Delete the generated **EventsAndDrawing.cpp** file.

Update the title.

3.4.1. Click Rectangle

Generate and draw a random filled rectangle that does not change its position nor its dimension each frame. Its minimum size is 60 x 40 pixels. It never is spawned closer than 50 pixels from the window borders. (re-use part of your code of previous lab). Use the color struct in structs.h to hold its color. The color of the rectangle is gray. When you click inside the rectangle, its color turns green. Clicking the green rectangle makes its size and position regenerate (random) and sets its color back to gray.

Tips:

- Use functions when appropriate.
- Enumeration for the rectangle state might come in handy.
- A global boolean or enum variable that is updated in the event functions and checked in the update/draw functions is how to do it.

3.4.2. Running Rectangle

The rectangle from previous exercise starts moving to the right when the ‘r’ key is pressed once. Each pixel that leaves the window on the right, appears again on the left edge of the window. When the ‘r’ key is pressed while it is moving, it stops. Clicking it still does as described as in previous exercise.

3.4.3. Bouncing line

The l key toggles the moving line on or off, it bounces off the edges of the window. Look for “bouncing line screen saver” videos. You implement just one line. When turned off, its position if frozen. It continues where it left off when turned on.

3.5. Explosion

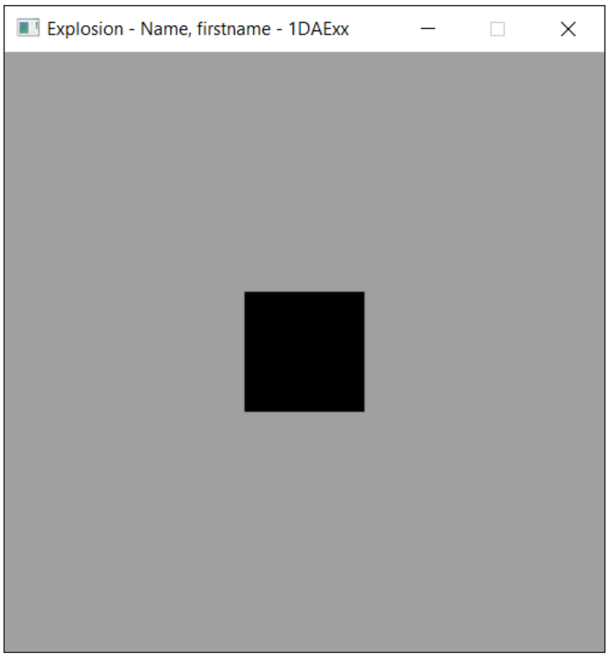
Create a new SDL project with name **Explosion** in your **1DAExx_04_name_firstname** folder.

Delete the generated **Explosion.cpp** file.

Update the title.

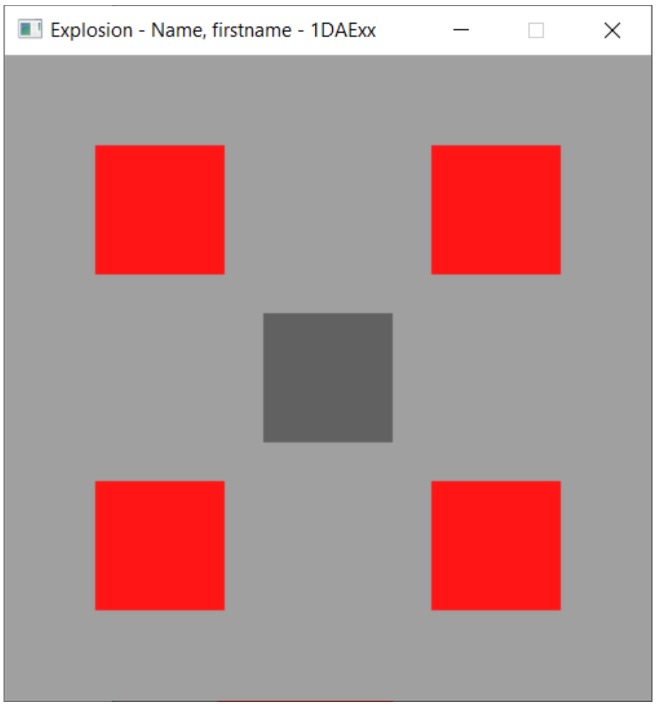
Give the window the same width and height.

In this exercise a bomb – a filled black square - is located in the center of the window.



When clicking with the left mouse button (`SDL_BUTTON_LEFT)` within the boundaries of the bomb it explodes, it becomes partly transparent and 4 red filled rectangles start moving towards the corners of the window.

When they leave the window a new bomb appears.



Make your code readable, create functions, e.g.:

- **ProcessMouseClick**: that checks whether the left mouse click happens within the bomb boundaries and starts the exploding when this is the case. This function should be called when an SDL_MOUSEBUTTONUP event occurs for the left mouse button (SDL_BUTTON_LEFT).

- **UpdateExplosion**: that changes the location of the explosion squares on the window. Call this function from within the Update function.

- **DrawExplosion**: Draws the bomb and the explosion squares. You call it in the Draw method.

4. Submission instructions

You have to upload the folder *1DAExx_04_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.

Make sure that you answered the quiz.

Compress this *1DAExx_04_name_firstname* folder and upload it before the start of the first lab next week.

5. References

5.1. Input/output manipulators

5.1.1. Operator Precedence

http://en.cppreference.com/w/cpp/language/operator_precedence

5.2. Leap year

https://en.wikipedia.org/wiki/Leap_year#Algorithm

5.3. Enumeration

https://www.youtube.com/watch?v=Pxvvr5FAWxg

http://www.learncpp.com/cpp-tutorial/4-5a-enum-classes/

5.4. Structs

http://www.learncpp.com/cpp-tutorial/47-structs/

5.5. ASCII table

http://www.learncpp.com/cpp-tutorial/27-chars/

5.6. Algorithm to draw circles and ellipses

http://www.mathopenref.com/coordcirclealgorithm.html

5.7. SDL events

5.7.1. Event driven programming

http://lazyfoo.net/tutorials/SDL/03_event_driven_programming/index.php

5.7.2. SDL event

https://wiki.libsdl.org/SDL_Event

5.7.3. Keyboard button event

https://wiki.libsdl.org/SDL_KeyboardEvent

5.7.4. SDL Keycode Lookup Table

https://wiki.libsdl.org/SDLKeycodeLookup

5.7.5. Mouse button event

https://wiki.libsdl.org/SDL_MouseButtonEvent

5.7.6. Mouse motion event

https://wiki.libsdl.org/SDL_MouseMotionEvent