Part 1. C++
 - Create “Hello world” program
 - Add static 2-dimensional integer array int A1[30][30] and dynamic array int* A2 = new int[30*30]
 - Fill A1 with random values (stdlib.h::rand()) using two for loops (0<i<30; 0<j<30)
 - Copy elements from A1 to A2 using for loop (0<i<30*30)
 - Create simple console menu using do-while loop
 - Add function which will print on screen elements of A2 array in matrix like style

Part 2. Cellular Automata (CA)
 - Create header file CAbase.h and implement class CAbase which will include CA functionality.
 - Add constructor and destructor
 - Add necessary variables: 2 dynamic arrays for old and new state of CA universe and int Nx and Ny variables to store universe size
 - Add basic functions - to set/get Nx and Nx; - to resize universe; - to set/check cell of past/future by (x, y)
 - Implement rules of Conway's Game of Life for cell (x, y) as a function. Add function to run evolution for every cell.
 - Add CAbase.h as a header to your C++ file, create there object of CAbase class and test its functionality: create universe 30x30; fill it with a few “alive” cells; print universe on screen in matrix like mode; run evolution and print it again; create do-while based menu for this functions (0.Exit; 1.Add; 2.Pint; 3.Evolve etc.).

Part 3. Qt visualisation
 - Create Qt project (Qt widget application, if main.cpp, mainwindow.h, mainwindow.cpp will not be created automatically, add them)
 - Add GameWidget* game object to mainwindow and create gamewiget.h and gamewiget.cpp with class GameWidget : public QWidget
 - Open mainwindow.ui and configure an interface of your program:
 -> “QBoxLayout” allows to combine other elements inside of it and can be used as a field for drawing
 -> “QPushButton” - for Start/Stop and other commands
 -> “QComboBox” - to choose one option from the list
 -> “QSpinBox” - to resize universe and set time interval between evolutions
 - Include CAbase.h to gamewiget.h as the header file and use its functionality for CA calculations
 - Implement functions for visualisation of CA evolutions in gamewiget:
 -> Constructor - define variables; set timer interval (timer->setInterval(int)); set size of CA universe (ca1.ResetWorldSize(int,int)); connect timer with evolution function (connect(timer, SIGNAL(timeout()), this, SLOT(newGeneration()));)
 -> void GameWidget::**startGame** - run timer, connected to evolution function
 -> void GameWidget::**stopGame** - stop the timer
 -> void GameWidget::**clear** - clear CA universe for new game
 -> void GameWidget::**newGeneration** - function for running the evolution and updating (redrawing) the game field
 -> void GameWidget::***mousePressEvent*** - standard function to estimate reaction

on mouse (set alive cells of CA)
 -> void GameWidget::***mouseMoveEvent*** - for mouse moving with hold button (multiple set alive cells)
 -> void GameWidget::**paintGrid** - prepare game field in QBoxLayout which you use for visualisation of universe state (draw a grid)
 -> void GameWidget::**paintUniverse** - draw state of the universe in QBoxLayout
 - Set “Game of Life” evolution in **newGeneration** function and test the program
 - (opt) Add Save/Load of universe state functions to mainwindow and gamewiget

Part 4. New functions (noise, erosion, fractals)
 - Noise - simple pseudo-random noise generator, based on CA. Can be used to create random state of CA universe for other functions.
 -> Use new cell evolution function instead of “Game of Life” evolution
 -> Every cell has 5 neighbours (including itself before the evolution): (CENTRE), (UP), (DOWN), (RIGHT), (LEFT)
 -> New state of every cell is calculated by formula (NEW) = (CENTRE) AND (UP) XOR (DOWN) XOR (RIGHT) XOR (LEFT) XOR (CENTER)
 -> Implement cell evolution for noise generator into CAbase.h and add additional functionality into Qt project to use it
 - Erosion - kind of CA evolution which emulates ground erosion process - the action of surface processes that remove ground from some location on the Earth's crust
 -> Add new cell evolution function into CAbase.h, which realise the following rules:
 -> Every cell has 8 neighbours: (UP-LEFT), (UP), (UP-RIGHT), (RIGHT), (DOWN-RIGHT), (DOWN), (DOWN-LEFT), (LEFT)
 -> Active cell will be removed if it does not have neighbours at any side: ((UP-LEFT) || (UP) || (UP-RIGHT)) && ((UP-RIGHT) || (RIGHT) || (DOWN-RIGHT)) && ((DOWN-RIGHT) || (DOWN) || (DOWN-LEFT)) && ((DOWN-LEFT) || (LEFT) || (UP-LEFT))
 -> Otherwise, cell will stays alive
 - Unlimited Life, Limited Life, Fractals - kind of CA evolution which use different rules of “Game of Life”
 -> Add new cell evolution function into CAbase.h, which realise the following rules:
 -> Unlimited Life — Unlimited growth case: The cell becomes alive if at least one of its neighbours is alive. The cell, once alive, remains alive forever.
 -> Limited Life — Limited growth case with Moore's neighbourhood (northWestCell, northCell, northEastCell, westCell, eastCell, southWestCell, southCell, southEastCell). The cell becomes alive if ONLY one of its neighbours is alive. The cell, once alive, remains alive forever.
 -> Fractals — Limited growth case with von Neumann neighbourhood(northCell, westCell, eastCell, southCell).
 - Add these functions to run evolution for every cell.

Part 5. Liquids and gases
 - Two liquids - evolution rules, which allow to emulate separation of two liquids with different surface tension.
 -> 1st liquid - "0"; 2nd liquid - "1"
 -> For every cell calculate sum of values in 9 neighbouring cells including itself:

(CENTRE), (UP-LEFT), (UP), (UP-RIGHT), (RIGHT), (DOWN-RIGHT), (DOWN), (DOWN-LEFT), (LEFT)
-> If sum > 5 of sum = 4 => set cell as "1", else "0"
-> Implement new type of cell evolution in CAbase.h and add functionality into Qt project to use it
- Two gases diffusion
-> Implement cell and universe evolution functions for gas diffusion with rules based on Margolus neighbourhood
-> Margolus neighbourhood in 2-dimensional case - 2x2 square block. CA divides universe on such blocks and change every block independently. CA works in 2 streps: 1.Margolus neighbourhood starts from (X, Y); 2.Starts from (X+1, Y+1).
-> Because of Margolus neighbourhood, you will need new universe evolution function, which include the following steps: Evolution1(universe->universeNew) -> CopyBack(uniNew->uni) -> Evolution2(uni->uniNew) -> CopyBack(uniNew->uni)
-> 1st gas - "0", 2nd gas - "1"
-> Cell evolution rules are the following: cell values inside of every Margolus neighbourhood shifts clockwise or counterclockwise. Direction of shift is chosen randomly for every algorithm step end every Margolus neighbourhood

Part 6. Snake game
- CA can be used as a base for some simple games. Next task is to make snake game with CA.
- Snake include “Head”, “Body” and “Tail”
-> Head - first cell of the snake, which sets the direction of moving. Different directions can be estimated by number in universe cell, where the Head located. At every evolution step, snake will moving to the next cell in chosen direction.
-> Body - main part of the snake. Within evolution, first element of the body will shifted to ex-Head cell, other elements - to the places of previous elements. (Use special numbers for body cells)
-> Tail - last cell of the snake. After evolution it moves to the last cell of the body and becomes empty.
- Additionally, you will need special number for “Food” cells and special function to create Food in random place of universe, after previous one was eaten by the snake. (Use special number)
- Special rule: evolution can change only ‘this’ (x, y) cell using information from neighbouring cells. Except the case, when the snake grows up.
- Implement functions for cell and universe evolution in CAbase.h
- Add special functions in gamewiget to process signals from keyboard (turn the snake up/down/left/right):
-> void GameWidget::**focusOnKeyboard** - change focus to the keyboard commands for some element of interface
-> void GameWidget::***keyPressEvent*** - list of reactions for different keys.

Part 7. Predator and victim
- There are three types of cells:
a) Food cell. Non-moving. The lifetime is infinite
b) Victim cell. Random Moving, has the lifetime

c) Predator cell. Random Moving, has the lifetime

- Rules:

-> If there is a predator in the current cell and a victim is in the neighbouring cell: The Predator moves to the cell with the victim

-> If there is a victim in the current cell and a predator is in the neighbouring cell: The victim doesn't move

-> If there is a victim in the current cell and a food is in the neighbouring cell and a predator is NOT in the neighbouring cell: The victim moves to the cell with the food

-> Each cell is reflected from the grid borders

-> Moving cells have a lifetime - the number of steps after which the cell dies

- Add new cell evolution functions into CAbase.h, which realise the following rules:

1. Set direction of move for every alive cell
    1. Check that the cell is alive or not
    2. Check if there is a predator in the current cell and a victim is in the neighbouring cell
    3. Check if there is a victim in the current cell and a food is in the neighbouring cell
    4. => Set the cell move on random direction
2. Universe evolution for every cell (move or stand, live or die, etc)
    1. Set the lifetime of cell to one less
    2. Check that the neighbouring cell has direction on current cell
    3. Random selection of a neighbour
    4. Move the selected neighbour
    5. Delete previous information about the selected neighbour