ASSIGNMENT NO.1

Statement

Develop a console-based car racing game in C++. The game should feature two cars moving on a racing track, and each car's position and points should be stored in a structure. The player can control each car through the keyboard, and the game loop should continuously update the display based on user input.

The game should include a scoring system where each car accumulates points during the race. Your code should provide a way for the player to quit.

Requirements: (These are just broad guidelines, feel free to use your own logic):

- Define a structure named Car that includes the following members:
    - position: an integer representing the car's position on the track.
    - points: an integer representing the accumulated points for the car.
- Use a nested structure named Racing Track that contains the following members:
    - Track Length: an integer representing the length of the racing track.
- Two instances of the Car structure to represent Car 1 and Car 2.
- Display a racing track with a specified length (e.g., 30 characters).
- Allow the user to control each car's movement using specific keys (e.g., 'a' and 'd' for Car 1, 'j'
- and 'l' for Car 2).
- Ensure that cars cannot move beyond the track boundaries.
- Implement a scoring system where each car accumulates points during the race.
- Display the current position and points of each car during the game.
- Implement a quit functionality (e.g., pressing 'q') to exit the game.
- Use a game loop to continuously update and display the game state.

Submission Guidelines:

• Implement the game logic using the provided structures and nested structures.

• Include comments in the code to explain the logic and functionalities.

• Provide a brief README file explaining how to compile and run the program.

• Feel free to consider additional features or improvements to enhance the gaming experience.

ASSIGNMENT NO.2

Statement:

"Strategic Five" stands as a timeless testament to the prowess of strategic thinking. With origins rooted deep in ancient civilizations, its allure transcends generations. As players place their pieces upon the board, they enter a cerebral battlefield, where each move carries the weight of calculated deliberation. Every placement is a gambit, a maneuver aimed at either outwitting the opponent or safeguarding against cunning traps. With each turn, the tension builds, the anticipation mounts, and the game evolves into a thrilling duel of intellects. Whether engaged in a leisurely match on a lazy afternoon or locked in fierce competition during a heated tournament, "Strategic Five" never fails to enthrall. Its magnetic pull leaves players yearning for just one more game, one more opportunity to showcase their strategic acumen and emerge victorious on the grid. Now, in this assignment, you are tasked with translating the essence of "Strategic Five" into C++ code. The game you craft must encompass the following features:

- The option for the board to assume dimensions of either 9x9 or 15x15, providing users with the
- flexibility to choose the size that suits their preference.
- Two distinct modes of play: Human vs Human and Human vs Bot.
- In implementing the game logic, ensure that the Bot, representing the computer opponent,
- exhibits logical prowess. While randomness may play a role in its decision-making, strategic
- algorithms should guide its moves, presenting human players with a formidable challenge.
- Whether facing off against a fellow human or testing their mettle against the machine, players
- should find themselves immersed in a captivating experience that truly embodies the essence of "Strategic Five."

Note: Your solution MUST be based on Classes.

ASSIGNMENT NO.3

Spinny Donut Assignment:

Write a C++ program to create a spinning donut, similar to this (NiceInsidiousGrub□size_restricted.gif (384×326) (gfycat.com). You need to define a Donut class which calculates all the pixel points and then draws the donut onto the console.

The following steps would guide you towards figuring out the solution for this.

1. Create a circle of radius R1 centered at R2.

Before thinking about how to spin the donut, we need to figure out how to draw a static donut first. So we have a circle of radius R1 centered at point(R2,0,0), drawn on the xy-plane. We can draw this by sweeping an angle — θ — from 0 to 2π.

2. Create a donut by rotating around the Y axis.

Now we take that circle and rotate it around the y-axis by another angle —let's call it ϕ, using the standard technique i.e. multiply by a rotation matrix. So, if we take the previous points and rotate about the y-axis.

3. Spin the donut around the X and Z axis.

We also want the whole donut to spin around on at least two more axes for the animation. Let's denote the rotation about the x-axis by A and the rotation about the z-axis by B. This can be achieved using: All of the above equations can be boiled down to a simple calculation through the power of Algebra, so instead of calculating the above values using matrix multiplications (which you can still do if you feel like it), you can use the below equations to get x, y, $z$points for a rotating donut. Try to reuse variables wherever possible to reduce computation and memory overhead. To achieve the rotating effect, you can clear the screen and draw the donut again and again using different values of A and B, where A and B are very slightly incremented (by a few decimal points).

4. Project donut onto 2D screen.

Up to this point, we have been calculating the values of the donut in 3D space, but to show the donut on a screen (which as we know is a 2D environment), we need to project those 3D points into 2D space. We can use the following equation for 3D to 2D projection: Where $K1$controls the scale (which depends on your pixel resolution) and $K2$controls the distance from the viewer to the donut. You can try out different values for these two to see what works best for your

configurations. For a screen_size of around 25, I'd recommend $K1$ =(10 . . . 15) and $K2$ = 5, but it will change depending on your implementation and screen_size.

5. Determine illumination by calculating surface normal.

Now that we know where to put the pixel, we still haven't considered which shade to plot. To calculate illumination, we need to know the surface normal — the direction perpendicular to the surface at each point. If we have that, then we can take the dot product of the surface normal with the light direction, which we can choose arbitrarily. That gives us the cosine of

the angle between the light direction and the surface direction: If the dot product is >0, the surface is facing the light and if it's <0, it faces away from the light. The higher the value, the more light falls on the surface. Luminance value could be calculated using: which ranges from -sqrt(2) to +sqrt(2). If it's < 0, the surface is pointing away from us, so we won't bother trying to plot it. Otherwise, we will plot it according to the luminance intensity, using the following characters

".,-~:;=!*#$@"

such that the higher the luminance, the higher index character will be used. Multiplying the luminance value by 8 would give us a value between 0 and 11, which can be used to index into the output characters.

Note: Make sure to read the task description carefully and do as require.

ASSIGNMENT NO. 4

In this assignment, you will explore the concept of composition in C++. Composition is a fundamental principle of object-oriented programming (OOP) that allows you to create complex objects by combining simpler ones. You will create a program that demonstrates the use of composition to model real-world relationships between classes.

Requirements:

- Create two classes: Engine and Car.
- The Engine class should have the following private member variables:
- int cylinders: the number of cylinders in the engine.
- int horsepower: the horsepower of the engine.

    - The Engine class should have a constructor that initializes the cylinders and horsepower variables.
- The Car class should have the following private member variables:
- string make: the make of the car (e.g., Ford, Toyota).
- string model: the model of the car (e.g., Mustang, Camry).
- Engine engine: an instance of the Engine class.
- The Car class should have a constructor that initializes the make, model, and engine variables.
- Implement getter and setter functions for the make, model, cylinders, and horsepower variables.
- Create a function in the Car class called printInfo that prints out the make, model, cylinders, and horsepower of the car.

Instructions:

- Implement the Engine and Car classes according to the requirements.
- Create an instance of the Car class in the main function and initialize it with appropriate
- values.
- Use the getter and setter functions to modify the make, model, cylinders, and
- horsepower of the car.
- Call the printInfo function to display the information about the car.

ASSIGNMENT NO. 5

Problem Statement

Description

In this assignment you are required to implement a class which will encapsulate C++ strings. This class will contain member functions to manipulate and access the string in various ways. A great deal of functionality will be implemented using C++ operator overloading.

Requirements

1. The character array (which is used to store the string) should be dynamic in nature.
2. Assume a default size of 50 bytes.
3. Your class must have the following constructors:

a) Default constructor
b) Constructor which accepts a character array
c) Constructor which takes the size of string

4. Implement class destructor.
5. Function which returns the size of string.
6. Function which returns the size of character array.
7. Overload all the operators, mentioned in Table 1.
8. In main() function, test all operations/functions of the string class. Print string(s) before
9. and after every operation.

Make the following string functions:

+ String concatenation

+= String concatenation a += b;

= = Test string equality bool

!= Test string inequality if(a != b) // Condition true

> String comparison if(a > b) // Condition true

< String comparison if(a < b) // Condition false

( ) Get sub-string

<< Print string using ostream cout << a << b;

>> Input string using istream cin >> a;

= String assignment b = a;

<< String left shift CMyString c;

>> String right shift CMyString c;

Submission

Submit your assignment via MS Teams

ASSIGNMENT NO. 6

Problem Statement:

Consider a group of sparse polynomials to be added as follows:

P1 = 0 + x + 0x2 + x3 + 0x4 + x5 + 0x6 + x7 + x8

P2 = 1 + x + x2 + 0x3 + x4 + 0x5 + x6 + 0x7 + x8

P3 = 0 + 0x + x2 + 0x3 + 0x4 + x5 + x6 + 0x7 + x8

P4 = 1 + 0x + x2 + x3 + 0x4 + 0x5 + 0x6 + x7 + 0x8

P5 = 0 + x + 0x2 + 0x3 + 0x4 + 0x5 + x6 + 0x7 + x8

The above set of polynomials will be given in the form of a text file (matrix) named example.txt with

contents as follows:

10

5

0 0 0 1 0

1 0 0 1 0

0 0 0 0 0

1 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 0

0 0 0 0 1

0 1 0 0 0

In the text file, the first line gives the number of polynomials, while the second line is the number of variables. The remaining numbers are the coefficient values for the polynomials. For the sake of simplicity, the coefficients are shown as binary, but they can be integer and reals

too. Code sparse polynomial addition technique that performs addition of a group of polynomials by reading corresponding data from a text file with the above given format. The data structures used to represent the polynomial will be a matrix (1D array).The polynomial P1 will be represented as follows:

0 0 0 1 0

Array data structure starting at 0 index

REQUIREMENT

You are required to code a GUI-based application that performs group addition as follows:

- Reads the data about the group of polynomials to be added from the text file.
- Provide an option to the user to display the contents of the file (with horizontal and vertical scrolling, required for large files if needed).
- Performs the addition, by first adding two polynomials and storing their result (in the data structure used), then adding the third polynomial to the result, and repeating this process till all polynomials have been added and then displaying the result on the screen and also writing in a text file named result_addition.txt. For the given text file, the first line will have the name of the input file i.e. example.txt and the second line will contain the coefficients of the result in their relative i.e. 2 3 3 2 1 2 3 2 4
- After all the polynomials have been added, display on the screen in the form of a message the (i) file name (ii) number of polynomials and variables (iii) number of non-zero values (iv) data structure used (v) and the total time taken for addition.
- The application is to be tested on the group of files provided.
- Please avoid the use of any built-in functions.

- Sample input files are separately attached with this problem statement.