Fri, 20 November

Geometric Shapes Calculation with Implemented Concepts in C++

We start with the #include <iostream> line which essentially tells the compiler to copy the code from the iostream file (which is used for managing input and output streams) and paste it in our source file.

Header iostream, that allows performing standard input and output operations, such as writing the output to the screen. Lines beginning with a hash sign **(#)** are directives read and interpreted by what is known as the preprocessor.

The same goes for the <cmath> and <cstring> files, then, we declare a function called main with the return type of integer, **main()** is the entry point of our program.

Whenever we run a C++ program, we start with the main function and begin execution from the first line within this function and keep executing each line till we reach the end.

We start a block using the curly brace **({)** here. This marks the beginning of main's function definition, and the closing brace **(})** marks its end.

All statements between these braces are the function's body that defines what happens when main is called.

After that, we use **"namespace std"**, to suffice us from writing **"std::"** for every input/output line, next, we create a class named "Shape", along with access specifiers.

We use public for functions that needs to be accessed anywhere in the code, and with "virtual" member class, we force the compiler to pick the method implementation defined in the object's class.

Then we define variables as float, with void so it doesn't return a value, next line, "protected" class will be defined, which is accessible from any class inside the code only.

Defining pi next, as a constant double fraction, then we close the "Shape" class, and head towards another class, Circumference, to handle the second output for the program.

This class is quite like the other "Shape" class, as it holds the same functions, only with different variable names, pi is also implemented.

Next step, we create new classes for the Geometric Shapes we're going to use, taking functions from the public Circumference class, adding an equation for the return, which is what we'll call later after inserting values.

We do the same step with "Shape" class, then we'll start a new code where we'll define a character that is used to collect user input data.

Forwards, we define variables as float along with defining pi in the next line, then we start main(void) block, it's not different than main(), except for it is more recommended to use, as it declares that it can only be called without any parameters.

Following, we use a "**loop**" to allow multiple execution of a block code, til a particular condition is satisfied, then we save classes of shapes in a variable that can be called for later outputs.

Successively, we print a hint for the user to know what to type, and what geometric shapes are supported by this program, ahead with a printed out message asking for user's choice of shape.

If condition was quite handy in this code, with it, and the help of <cstring> library, we can track user input to find out what did he wrote, parallel with another questions about the chosen shape, to use it with equations and solve the puzzle.

Thereafter, as soon as we finish with shapes' list, we use "**else**" to point the user into a message informing him about his wrong choice, looming with "goto loop" to re-start all over again.

Latterly, we merge user input values with **class** variables, getting it to the final stage of calculating, twenty lines will be written to handle all the shapes confined in this code.

Semi-final step will be viewing the answer, we do that by simple "**cout**" code, and the final step will be **return**ing "0" to our main(void) code block, successfully illuminating the program.

To be noted, **Abstraction** concept implements **Encapsulation** concept within it's code, **Encapsulation** is found in C++ even before the existence of access specifiers, most new learners **doesn't even know that they're using it**.

Along with **other concepts** implemented inside the merely known programming, it confuses a wide range of students when teaching them these concepts **late**.