Creative Software Programming Practice (week-10-1)

Every assignment will be announced on **Thursday** and should be sumitted by next **Tuesday**.

We have practice classes on Wednesdays and Thursdays.
The contents of the practice class are different from the assignments and aim to be completed on the same day.

Assignments will be publish on Thursday and must be submitted by the next Tuesday.

In this week **Handed out will be Nov 5, 2020, Due Nov 10, 2020**

Topics

1. Practices-1
2. Practices-2

1. Practices-1

We learned about polymorphism and inheritance until the previous hour.
Therefore, we will practice implementing a program using this. Write a program that works as follows:

1. Define class Shape that has a constructor taking *width* and *height* as parameters (double type).
2. Define class **Circle**, **Triangle** and class **Rectangle** which inherit from class Shape. Each subclass also has a constructor taking *width* and *height* as parameters (double type).

assume that PI = 3.14

3. All classes have a member function, `double getArea()`.
4. `Shape::getArea()` is a pure virtual function.
5. Take inputs from the user and create objects of proper type according to the input by new operator, and then put those objects into `std::vector<Shape*> arr`.
6. When the user enters 0, call the getArea() of each element of arr to print out calculated area. Each element of arr must be deallocated after use.
7. Do not use the type casting operator throughout the code.
8. This program should take user input repeatedly

Input:

- r *width*, *height*

  create a rectangle

- t *width*, *height*

  create a triangle

- c *width*, *height*

  create a circle

- 0

  print the results and quit the program

Output:

Areas of the shapes

Example:

```
r 10 5
t 2 4
c 3 3
0
50
4
28.26
```

```
// shapes.h

class Shape {
public:
    double width, height;

    Shape(double width, double height)
    : width(width), height(height) {
    }

    virtual getArea() = 0;
};


class Rectangle ...
class Triangle ...
class Circle ...
```

```
// shape-main.cc

#include <iostream>

int main() {

}
```

Pratice-2

Write a program that works as follows:

1. Class **Animal** has two member variables, `std::string name` and `int age`, and has a constructor that initializes the values of name and age by taking a string and an integer as arguments.
2. Class **Zebra** that inherits **Animal** has a member variable, `int numStripes`, and has a constructor that initializes the values of name, age, and numStripes by taking a string, an integer, and an integer as arguments.
    - The member variable of the parent class is initialized through the constructor of the parent class.

3. Class **Cat** that inherits **Animal** has a member variable, `std::string favoriteToy`, and has a constructor that initializes the values of name, age, and favoriteToy by taking a string, an integer, and a string as arguments.
    - The member variable of the parent class is initialized through the constructor of the parent class.
4. Each class has a member function printInfo()
    - Animal::printInfo() does nothing. make it pure virtual function
    - Zebra::printInfo() and Cat::printInfo() print out the type, name, age, and number of stripes (or favorite toy) as shown in the following example.
5. Create **Zebra** or **Cat** objects according to user input, ant put them into `std::vector<Animal*>` animals.
6. When the user enters 0, call the printInfo() function of each element of animals. Each element of animals must be deallocated after use.
7. Do not use the type casting operator throughout the code.
8. This program should take user input repeatedly

Input:

- z *name*, *age*, *numStripes*
- c *name*, *age*, *favoritToy*
- 0

  print the results and quit the program

Output:

The result for calling printInfo() functions

```
// animal.h

class Animal ...
class Cat ...
class Zebra ...
```

```
// animal-main.cc

#include <iostream>

int main() {

}
```

```
```