Creative Software Programming Assignment#10 (week-10)

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

In this week **Handed out will be Nov 6, 2020, Due Nov 12, 2020**

1. Canvas

We are making a canvas where we can draw shapes. We want to draw **Point**, **Triangle**, **Rectangle** and **Circle** on the canvas.

First, there is a primitive type of `std::pair<float, float>` called Point_. `std::pair` is a type that takes two types, and is used here to indicate a point (*x*, *y*) or a size (*width*, *height*).
*The first element of `std::pair` can be accessed via **first** and the second element via **second.***

The class **Drawable** indicates that an object is a drawable object.

- **Drawable** has two variables as private, visible(`bool`) and center(`Point_`).
- **Drawable**'s visible determines whether or not the object is visible when it is drawn.
- **Drawable** offset indicates the object's position (*x*, *y*).
- **Drawable** has getter and setter for visible and offset.
  - `void set_offset(const Point_& offset)`
  - `const Point_& get_offset() const`
  - `void set_visible(bool visible = true)`
  - `bool get_visible() const`
- **Drawable**'s draw returns `std::vector<Point_>`. This is a vector containing all the points to be drawn.
  - *That is, a rectangle with offset (2, 2) and size (3, 3) should return {(2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4), (4, 2), (4, 3), (4, 4)}*

The class **Fillable** indicates which object has a size.

- **Fillable** has two variables as private, fill(`bool`) and size(`Point_`).
- **Fillable**'s fill determines whether to fill the drawable's fill object when it is drawn. If **false**, only the border should be drawn without filling the inside.
  - The borader mean boundary of each row (so, only two or one point in each row)
- **Fillable** size indicates the size of the object (*width*, *height*).
- **Fillable** has getter and setter for fill size.
  - `void set_fill(bool fill = true)`
  - `bool get_fill() const`
  - `void set_size(const Point_& size)`
  - `const Point_& get_size() const`

The class **Canvas** can take a Drawable object and draw it.

- **Canvas** initially receives size (`size_t row, size_t col`).
- You can change the size with `void resize(size_t row, size_t col)`.

- You can add a **Drawable** object through `size_t add(T* component)`.
    - This function returns the *id* of an object. You can access it later via the object's id.
- Clears all **Drawable** objects registered through `void clear()`. The object should be deleted.
- Print all components registered through `void draw()`.
- Empty space is indicated by `.` and space to be filled is indicated by `ch(*)`. ch also has getter and setter.
- This class can be accessed through `Drawable* at_drawable(size_t index)` and `Fillable* at_fillable(size_t index)`. So, `canvas.at_drawable` gets the 3rd element of the canvas.
- When this object is destroyed, all registered objects must be properly released.

Assignment Structure

- week-10
    - shape.h
    - main.cc
- The input is given as:
    - Initially, canvas sizes w and h are given.
    - add {*point*|*rectangle*|*circle*|*triangle*} *x y* {*w*, *h*|*size*}
    - draw // print all registered elements through draw.
    - set *id* {*fill*|*visible*} {*true*|*false*}
    - set *id* size *w h* (or just size if circle)
    - set *id* offset *x y*
    - clear
    - exit for exit program

```
4 4
add point 1 1
add rectangle 2 2 3 3
draw
clear
add point 1 1
set 0 visible false
draw
clear
add triangle 3 3 2 3
draw
clear
add circle 1 1 3
draw
exit
```

- The output is as follows:

```
0
1
*...
.***
.***
.***
```

```
0
....
....
....
....
0
..*.
.***
.***
....
0
****
****
***.
*...
```

- **PI is 3.141592f**
- **When printing after all calculations, the position is determined through rounding.**
- **Rectangle offset is top-left point of rectangle**
- **The offset of circle is center**
- **The offset of the triangle is the center of the lower side. That is, the vertices are above h as offset and next to $\pm\frac{w}{2}$.**

```cpp
#include <utility>
#include <vector>
#include <functional>
#include <iostream>
#include <cmath>

using Point_ = std::pair<float, float>;

class Drawable {
public:
    Drawable(Point_ offset = { 0, 0 }, bool visible = true)
        : offset(offset), visible(visible) {
    }

    virtual std::vector<Point_> draw() = 0;

    void set_offset(const Point_& offset) {
        this->offset = offset;
    }
    const Point_& get_offset() const {
        return offset;
    }

    void set_visible(bool visible = true) {
        this->visible = visible;
    }
    bool get_visible() const {
        return visible;
    }

private:
    bool visible;
```

```
    Point_ offset;
};

class Fillable {
public:
    Fillable(Point_ size, bool fill = true)
        : size(size), fill(fill) {
    }

    void set_fill(bool fill = true) {
        this->fill = fill;
    }
    bool get_fill() const {
        return fill;
    }

    void set_size(const Point_& size) {
        this->size = size;
    }
    const Point_& get_size() const {
        return size;
    }
private:
    bool fill;
    Point_ size;
};

class Point : public Drawable {
public:
    Point(Point_ offset = { 0, 0 }, bool visible = true)
        : Drawable(offset, visible) {
    }

    std::vector<Point_> draw() {
        return { get_offset() };
    }
};

class Rectangle : public Drawable, public Fillable {
public:
    Rectangle(Point_ offset, Point_ size, bool fill = true, bool visible = true)
        : Drawable(offset, visible), Fillable(size, fill) {
    }

    std::vector<Point_> draw() {
    }
};

class Circle : public Drawable, public Fillable {
public:
    Circle(Point_ offset, size_t size, bool fill = true, bool visible = true)
        : Drawable(offset, visible), Fillable({ size, size }, fill) {
    }

    std::vector<Point_> draw() {
    }
};
```

```
class Triangle : public Drawable, public Fillable {
public:
    Triangle(Point_ offset, Point_ size, bool fill = true, bool visible = true)
        : Drawable(offset, visible), Fillable(size, fill) {
    }

    std::vector<Point_> draw() {
    }
};

class Canvas {
public:
    Canvas(size_t row, size_t col, char ch = '*')
        : row(row), col(col), ch(ch), matrix(row, std::vector<bool>(col, false))
{
    }
    ~Canvas() {}

    void resize(size_t row, size_t col) {
        row = row;
        col = col;
    }

    size_t add(Point* drawable) {
        drawable_components.push_back(drawable);
        fillable_components.push_back(nullptr);

        return drawable_components.size() - 1;
    }

    template <typename T>
    size_t add(T fillable) {
        drawable_components.push_back(fillable);
        fillable_components.push_back(fillable);

        return drawable_components.size() - 1;
    }

    void draw() {
    }

    void drawable_apply(const std::function<void(Drawable*)>& f) {
        for (auto component : drawable_components) {
            f(component);
        }
    }
    void fillable_apply(const std::function<void(Fillable*)>& f) {
        for (auto component : fillable_components) {
            f(component);
        }
    }

    void clear() {
        drawable_components.clear();
        fillable_components.clear();
    }

    void set_ch(char ch) {
```

```
        ch = ch;
    }
    char get_ch() const {
        return ch;
    }

    Drawable* at_drawable(size_t index) {
        return drawable_components[index];
    }
    Fillable* at_fillable(size_t index) {
        return fillable_components[index];
    }

private:
    size_t row, col;
    char ch;
    std::vector<Drawable*> drawable_components;
    std::vector<Fillable*> fillable_components;

    std::vector<std::vector<bool>> matrix;
};
```

```
```