EIE
Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

Estructuras Abstractas de Datos y Algoritmos para Ingeniería

Laboratorio 2
Herencia y Polimorfismo

Dennis Chavarría Soto
Jesus Zuñiga Méndez

II ciclo 2019

1. Introduccion

La herencia, en la programación orientada a objetos, es un mecanismo fundamental que es meritorio de atención, debido a las posibilidades que ofrece respecto la reutilización de código, además de simplificar y aumentar la comprensión del código. Es así, como este laboratorio tiene como eje aplicar los conocimientos adquiridos durante la clase magistral y ponerlos en práctica al resolver un problema dado, en este caso, un sistema que cree polígonos los cuales pueden tener caracteristicas similares tales como el perímetro, o el área.

2. Funcionamiento del programa

2.1. Archivo de includes

Este archivo contiene todas las instrucciones para incluir los archivos de cabecera que contienen definiciones de las clases, así como la del número PI. Su importancia radica en que permite asociar los archivos de cabecera que dependen de los .cpp del código.

2.2. Clase Principal

Todo lo referente a esta clase se encuentra en el archivo Tools.hpp. Esta clase es la que sirve para instanciar al objeto principal que se encarga de dar funcionamiento al programa. Consta de bastantes métodos fundamentales. Sin embargo, solo el constructor Principal() y el destructor carecen de gran lógica. En sí, no tienen lineas de código. Esta clase está contenida en el archivo Tools.hpp

2.2.1. Bienvenido()

Método genérico que tiene un grupo de çout"para imprimir una serie de lineas de texto al inicio de código. Es meramente decorativo. Destaca el uso de "FORMATO_ANSI_COLOR_X"para proporcionar detalles de color al texto mostrado en pantalla.

2.2.2. Menú()

Es genérico y se encarga de imprimir varias lineas de texto mediante çout", estas indican las instrucciones de operación del programa. En realidad no tiene una entrada de datos, pues, de ello se encargará la función main del código.

2.2.3. HacerArregloVertices(int cantidad, Vertice* arreglo)

HacerArregloVertices es un método genérico, tiene el propósito de preparar el arreglo de puntos que los polígonos usarán. Recibe un valor para la variable int cantidad, así como el puntero de tipo Vertice, arreglo. Tiene dos variables locales, x, y; estas guardarán los valores de la coordenada en la cuál se encontrarán localizados los vertices que el usuario ingresa. Se almacenarán los datos con çin". Finalmente, con el arreglo recibido, se itera entre un grupo de espacios para guardar lo que el usuario ingresó según corresponde. Nótese, los espacios son vertices, en realidad.

2.2.4. Color()

Método de tipo string que se encarga de cambiar la apariencia de determinado texto de la pantalla. Utiliza la variable respuesta, con un determinador color, luego nombre y color, a quienes

asigna un identificador asociado al color, luego con un ciclo "for", imprime el texto con el nuevo formato. El resultado de la respuesta de texto creada se retorna finalmente.

2.2.5. Instancias de diferentes tipos de polígonos

A continuación se explicará en qué consisten las instancias de cada polígono. Debe considerarse que el programa solicita, en la función main. Como consideración, existe una superclase base llamada figura, 3 subclases: triangulo, circulo, rectangulo; y tres especializaciones en forma de clases para el triangulo, dadas como isosceles, equilatero y escaleno.

- MtdCirculo: Define un arreglo de dos vertices. Luego se pasa como parámetro el arreglo al método apropiado para solicitar las coordenadas. Posteriormente se instancia un poligono de tipo circulo. Se tiene una variable local de tipo string llamada lectura, esta guadara el dato que se solicita a continuación (el nombre), así, se establece el nombre del círculo y se invoca el color que se estableció en el método color.

- MtdRectangulo: Define un arreglo de cuatro vertices. Luego se pasa como parámetro el arreglo al método apropiado para solicitar las coordenadas. Posteriormente se instancia un poligono de tipo rectangulo. Se tiene una variable local de tipo string llamada lectura, esta guadara el dato que se solicita a continuación (el nombre), así, se establece el nombre del rectangulo y se invoca el color que se estableció en el método color.

- MtdTriangulo; Define un arreglo de tres vertices. Luego se pasa como parámetro el arreglo al método apropiado para solicitar las coordenadas. Se tiene una variable local de tipo string llamada lectura, esta guadara el dato que se solicita a continuación (el nombre), así, se establece el nombre del triangulo y se invoca el color que se estableció en el método color. Ademas se comparan las distancias entre los vertices ingresados con el proposito de saber si se trata de un triangulo escaleno, equilatero o isosceles. Una vez obtenido el resultado determinante, se instancia un poligono de la subclase triangulo apropiada.

- MtdCualquierFigura: Igual que las instancias anteriores, se definirá un conjunto de vértices, no obstante, para este caso se solicitará la cantidad de ellos de forma específica al usuario, luego se instanciará de la misma forma que los otros casos, pero solo se usarán los métodos y atributos que contiene la clase Figura

2.3. Clase Vertice

La clase Vértice consiste, una vez instanciada, en un punto en el plano 2D, con atributos que indican su posición en el eje x, así como en el eje y. Estos consisten en variables de tipo float que permiten, además, calcular distancias entre otros vértices en el plano de dos dimensiones. Esta clase consta, también de un identificador que puede ser utilizado para iterar dentro de un arreglo de estos objetos, como se verá más adelante. Así mismo, esta cuenta con varios métodos, entre ellos: Vértice() como el constructor, Vertice() como el destructor; string operator () que permite mostrar una descripción del punto, sea sus coordenadas; double operator($constVertice\&rhs$), $esteconsiste, enrealidad, enlasobrecarga$

2.3.1. Funcionamiento del operador

Este operador está programado de forma que establece una variable local llamada respuesta, de tipo string, la cual será utilizada para almacenar el texto que será impreso para mostrar una descripción del vértice, sin embargo, requiere de las variables locales strX y strY y estas últimas

de la función to_string(x), to_string(y) respectivamente, con el propósito de cambiar el tipo de los valores que esas variables, contenidas en la función, tienen a un string. luego se añaden a respuesta y el resultado se retorna.

2.3.2. Funcionamiento del operador

Este operador recibe otro objeto de tipo vertice como parámetro; inicialmente se establece una variable local "distanciaçon el valor de 0, luego se calculan xcuadrado y su homólogo en y, mediante la diferencia entre el valor contenido en el atributo de la coordenada correspondiente y el del punto obtenido por referencia. posteriormente se elevan ambas variables al cuadrado mediante la función pow() de la librería cmath. Cuando se obtienen los valores de xcuadrado y ycuadrado, se suman y se obtiene la raíz cuadrada de estos mediante la función sqrt(), lo cual equivale a la distancia entre los puntos. Este valor se retorna.

2.4. Clase Figura

Esta es la clase base utilizada para los polígonos y define sus atributos básicos como el perímetrofig de tipo float, así como el área, de igual tipo. También tiene un string nombre y color. Incluye un identificador que se asigna con base en un identificarEstático que aumenta conforme se instancia una nueva figura. Se separa en un archivo .hpp con la definición de sus métodos y atributos y un .cpp con el respectivo código que permite funcionar lo anterior.

2.4.1. Metodo constructor y método destructor

Estos métodos no tienen lógica programada, actúan por defecto.

2.4.2. Metodo Virtual superficie

Define una superficie de un polígono a partir de la multiplicación de las coordenadas x, y que posee. Devuelve este resultado.

2.4.3. Metodo Virtual perimetro

Define un valor de perimetro dado al multiplicar por cuatro la coordenada x que posee. Devuelve este resultado.

2.4.4. Sobrecarga de operador

El operador no esperará nada más que la figura que está antes que el. Al invocársele, devuelve el nombre, color, superficie, perimétro.

2.5. Clase Circulo

Separada en dos archivos, un .hpp que contiene la definición de la clase con su atributo especial radioz los métodos provenientes de los metodos virtuales de figura, double perimetro y double superficie; para calcular el perimetro y el area respecticamente, luego, cuenta con el operador sobrecargado -", utilizado para mostrar todas las características de este polígono. Hereda de Figura sus atributos y métodos.

2.5.1. Metodo constructor

Establece un puntero de tipo vertice que apunte al arreglo de puntos que recibe; una vez hecho lo anterior, obtiene la distancia y la asigna como el valor del radio.

2.5.2. Metodo perimetro

Asigna al atributo perimetrofig el valor del perimetro obtenido al multiplicar por dos el numero PI por el radio. Este valor es devuelto.

2.5.3. Metodo superficie

Asigna al atributo area el valor de la superfice obtenido al multiplicar el numero PI por el radio elevado al cuadrado.

2.5.4. Sobrecarga de operador

El operador no esperará nada más que el circulo que está antes que el. Al invocársele, devuelve el nombre, color, superficie, perimétro y radio del círculo.

2.5.5. Metodo destructor

Como su nombre lo indica, unicamente actua como el destructor del objeto.

2.6. Clase Rectangulo

Esta clase tiene entre sus atributos básicos la base y la altura, ambos de tipo float. Se separa en un archivo .hpp con la definición de sus métodos y atributos y un .cpp con el respectivo código que permite funcionar lo anterior. Hereda de Figura sus atributos y métodos.

2.6.1. Metodo constructor

Establece un puntero de tipo vertice que apunte al arreglo de puntos que recibe; una vez hecho lo anterior, obtiene la distancia entre ellos y debe compararla en un ciclo, que, a base de condicionales, discrimina el lado de mayor extensión, y se centra en los dos restantes, comparando y determinando cual es el mayor y el menor para asignarlos como base o altura.

2.6.2. Metodo perimetro

Asigna al atributo perimetrofig el valor del perimetro obtenido al multiplicar por dos la base y sumarlo a la altura, también multiplicada por dos. Este valor es devuelto.

2.6.3. Metodo superficie

Asigna al atributo area el valor de la superfice obtenido al multiplicar la altura por la base. Este valor es retornado.

2.6.4. Sobrecarga de operador

El operador no esperará nada más que el rectangulo que está antes que el. Al invocársele, devuelve el nombre, color, superficie, perimétro, base y altura del rectangulo.

2.7. Clase Triangulo

Esta clase tiene entre sus atributos básicos lado1, lado2, lado3, todos de tipo float. Se separa en un archivo .hpp con la definición de sus métodos y atributos y un .cpp con el respectivo código que permite funcionar lo anterior. Hereda de Figura sus atributos y métodos.

2.7.1. Metodo constructor y destructor

Estos métodos no tienen lógica programada, actúan por defecto.

2.7.2. Sobrecarga de operador

El operador no esperará nada más que el triangulo que está antes que el. Al invocársele, devuelve el nombre, color, superficie, perimétro, base y altura del triangulo.

2.8. Especializaciones de triangulo

Los siguientes archivos están contenidos en un mismo archivo .cpp y .hpp que reciben el nombre de Triangulo.cpp y Triangulo.hpp

- Equilatero: Se separa en un archivo .hpp con la definición de sus métodos y atributos y un .cpp con el respectivo código que permite funcionar lo anterior. Hereda de Figura y triangulo sus atributos y métodos; del último la longitud de los lados.
- Isosceles: Se separa en un archivo .hpp con la definición de sus métodos y atributos y un .cpp con el respectivo código que permite funcionar lo anterior. Hereda de Figura y triangulo sus atributos y métodos; del último la longitud de los lados.
- Escaleno: Esta clase tiene entre sus atributos básicos el semiperimetro s. Se separa en un archivo .hpp con la definición de sus métodos y atributos y un .cpp con el respectivo código que permite funcionar lo anterior. Hereda de Figura y triangulo sus atributos y métodos; del último la longitud de los lados. enditemize

2.8.1. Sobrecarga de operador

Las tres especializaciones anteriores tienen este mismo operador; no esperará nada más que la especialización que está antes que el. Al invocársele, devuelve el nombre, color, superficie, perimétro, base y altura del triangulo.

2.8.2. Metodo constructor y destructor

El método constructor recibe como parámetro un puntero de tipo vertice que apunta a los vertices especificados por él, con sus coordenadas establecidas en el respectivo método. para las tres especializaciones anteriores, se se asignan de igual forma a los lados los valores de la distancia entre cada punto. Para el caso del isosceles, se usa un algoritmo muy similar al del rectangulo, para utilizar los lados adecuados para la base y la altura, solo que, en este caso se debe asignar un tercer valor que corresponde a la hipotenusa.

2.8.3. Sobrecarga de operador

Las tres especializaciones anteriores tienen este mismo operador; no esperará nada más que la especialización que está antes que el. Al invocársele, devuelve el nombre, color, superficie, perimétro, base y altura del triangulo.

2.8.4. Superficie de Isosceles

Se multiplica la base (lado1) por la altura (lado2) y se devuelve el resultado.

2.8.5. Superficie de Escaleno

Se utiliza la formula (sqrt(abs(semiperimetro * (s-lado1) * (s-lado2) * (s-¿lado3))) para calcular el area, donde "sçorresponde al semiperimetro.

2.8.6. Superficie de Escaleno

Se utiliza la formula sqrt(3)/2) * pow(this-¿lado1,2) y se devuelve el resultado

2.8.7. Perimetro de los triangulos

Las especializaciones recurren al metodo que tiene la clase triangulo para calcular su perimetro, la formula consiste en lado1+lado2+lado3. El resultado debe retornarse.

2.9. Clase impresora

Consta solo de un archivo .cpp, y uno .hpp Tiene un método constructor que obtiene las variables para asignar colores e imprimir en la pantalla, También tiene su método destructor

2.9.1. Metodo constructor y método destructor

Estos métodos no tienen lógica programada, actúan por defecto.

2.9.2. ObtenerColor(string nomColor)

Método de tipo string, tiene como variables locales tamanio, de tipo entero, con un valor inicial igual que 6; luego dos arreglos y una variable, todos de tipo string. Un arreglo es llamado tamanio y contiene los colores; por otra parte, color contiene las "fórmulas"que permiten que los colores funcionen. Por medio de un ciclo "for"se imprime el contenido de respuesta, basado en las características de color que se implementaron. Este resultado se devuelve

2.9.3. imprimirArchivo(const T& objeto, string rutaArchivo)

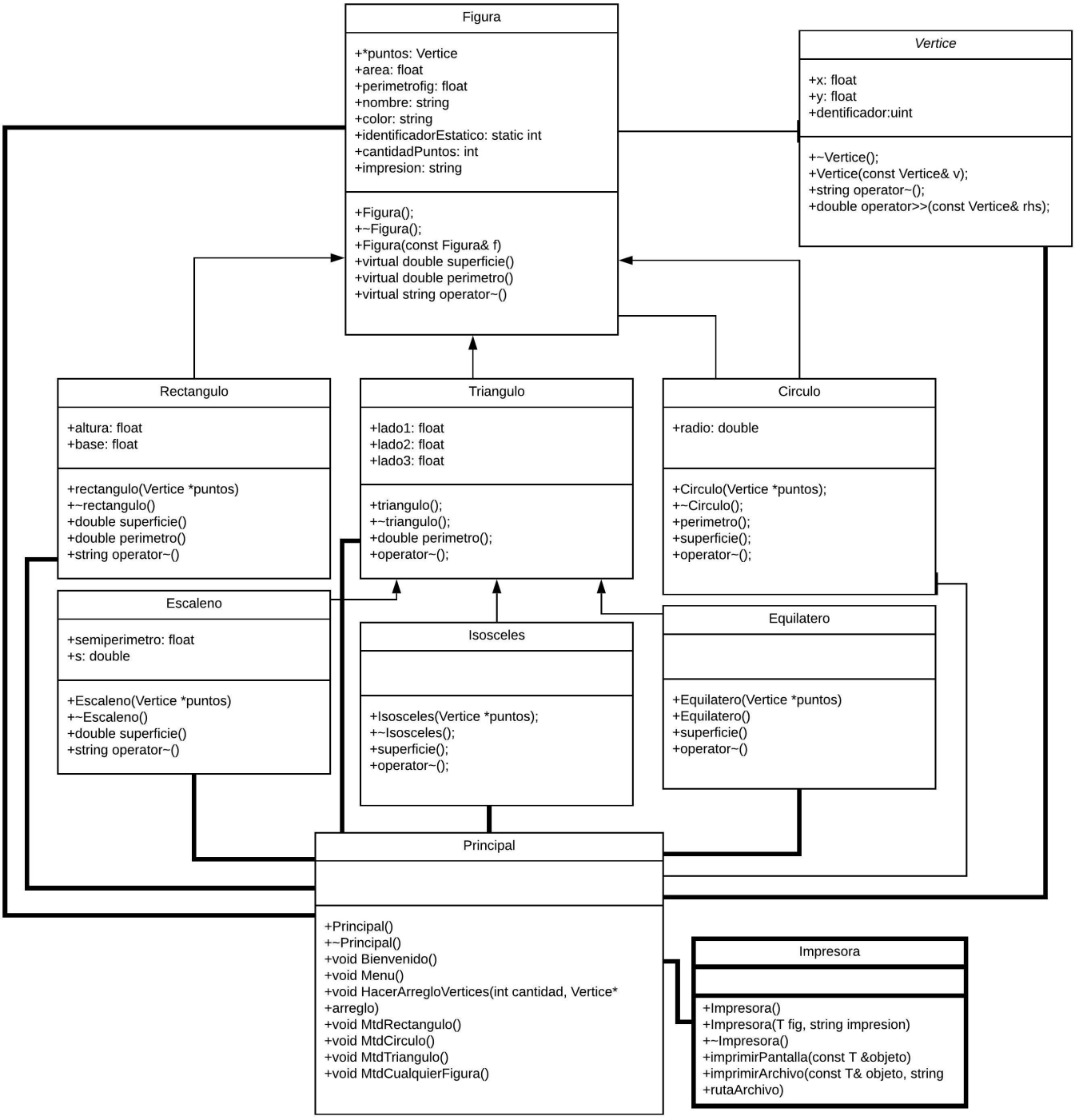
Método genérico que se encarga de manejar un archivo, consta de un condicional que avisa en caso de no encontrarlo. Su propósito es obtener los parámetros de este para poder imprimir lo que se solicita con el formato establecido, colores, alineación. SUs parámetros son una referencia de un objeto cualquiera, así como un string que especifica una ruta a un archivo.

3. Conclusiones

- Desarrollo de un programa con la capacidad de trabajar con polígonos de dos dimensiones, calcular su perímetro, superficie y manejar otros parámetros relacionados.
- Aplicación satisfactoria de la herencia para la reutilización de código y simplificación del programa
- De forma satisfactoria, se implementaron los conocimientos de herencia y polimorfismo para completar los objetivos requeridos en el enunciado del laboratorio.
- Para la clase impresora se envía la impresión como parámetro, pues, al utilizar el operador sobrecargado , el programa no funcionaba, así pues, se utiliza esto como un método.

4. Anexos

4.1. Diagrama de clases



4.2. Vertice.cpp

```
#include "../include/Includes.hpp"
using namespace std;

/**
 * @brief constructor por defecto
 */
Vertice::Vertice(){
}
/**
 * @brief desconstructor por defecto
 */
Vertice::~Vertice(){
}
/**
 * @brief constructor por copia
 * @param v es un objeto
 */
Vertice::Vertice(const Vertice& v){
    this->x=v.x;
    this->y=v.y;
    this->identificador = v.identificador;
}
/**
 * @brief sobrecarga operador ~ para convertir a texto
 */
string Vertice::operator~(){
    string respuesta = "";
    string strX = to_string(x);
    string strY = to_string(y);
    string identi = to_string(identificador);
    respuesta = ("punto: " + identi + " Coordenada X "+ strX +" coordenada Y "+
     strY);
    return respuesta;
}
/**
 * @brief sobrecarga operador ~ para convertir a texto
 */
double Vertice::operator>>(const Vertice& rhs){
    float distancia = 0;
    float xcuadrado = (this->x - rhs.x);
    float ycuadrado = (this->y - rhs.y);
    xcuadrado = pow(xcuadrado,2);
    ycuadrado = pow(ycuadrado,2);
    distancia = sqrt (xcuadrado + ycuadrado);
    return distancia;
}
```

4.3. Vertice.hpp

```
#ifndef VERTICE_H
#define VERTICE_H

#include "./Includes.hpp"

using namespace std;

/**
 * @brief clase que modela un punto en el espacio
 */
    class Vertice
    {
        public:
            Vertice();
            ~Vertice();
            Vertice(const Vertice& v);
            string operator~();
            double operator>>(const Vertice& rhs);
            float x;
            float y;
            int identificador;//uint
        //private:

    };
#endif
```

4.4. Figura.cpp

```
#include "../include/Includes.hpp"

using namespace std;

/**
 * @brief constructor por defecto
 */
Figura::Figura(){
}
/**
 * @brief destructor por defecto
 */
Figura::~Figura(){
}
/**
 * @brief constructor por copia
 */
Figura::Figura(const Figura& f){
    this->area = f.area;
    this->perimetrofig = f.perimetrofig;
    this->nombre = f.nombre;
    this->color = f.color;
    this->identificadorEstatico = f.identificadorEstatico;

}

/**
 * @brief metodo que calcula una superficie generica
 */
double Figura::superficie(){
    double superficie = this->x * this->y;
    return superficie;
}

/**
 * @brief metodo que calcula un perimetro generico
 */
double Figura::perimetro(){
    double perimetro = 4*x;
    return perimetro;
}

/**
 * @brief sobrecarga del operador ~
 */
string Figura::operator~(){
    string retorno;
    string strpuntos = "";
    for (int i=0; i < this->cantidadPuntos; i++){
        strpuntos = strpuntos + ~puntos[i];
        strpuntos = strpuntos + "\n";
    }
    retorno = strpuntos + "\n";
    retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
                "Mi color es "+this->color + " \n" +
                "Mi superficie es "+to_string(this->superficie())+ " \n" +
                "Mi perimetro es "+to_string(this->perimetro())+ " \n"
```

```
58                 );
59     return retorno;
60 };
```

4.5. Figura.hpp

```
#ifndef FIGURA_HPP
#define FIGURA_HPP

#include "./Includes.hpp"

using namespace std;

    /**
     * @brief clase que se encarga de crear una figura en 2D
     */
    class Figura : public Vertice
    {
        public:
            Figura();
            ~Figura();
            Figura(const Figura& f);
            Vertice *puntos;
            virtual double superficie()=0;
            virtual double perimetro()=0;
            virtual string operator~()=0;
            float area;
            float perimetrofig;
            string nombre;
            string color;
            static int identificadorEstatico;
            int cantidadPuntos;
            string impresion;
    };
#endif
```

4.6. Circulo.cpp

```
#include "../include/Includes.hpp"



/**
 * @brief Constructor de circulo
 * @param temp es un puntero de un arreglo de vertices
 */
Circulo::Circulo(Vertice *temp){
    this->puntos=temp;
    this->radio=puntos[0]>>puntos[1];
    this->perimetrofig = this->perimetro();
    this->area = this->superficie();
    cout<< "El radio es: " << this->radio<< endl;
    this->area = this->superficie();
    this->perimetrofig = this->perimetro();
    cout << "el area es: " << this->area << endl;
    cout << "el perimetro es: " << this ->perimetrofig << endl;
}

/**
 * @brief Calcula el perimetro
 */
double Circulo::perimetro(){
  this->perimetrofig = ( 2*PI*(this->radio));
  return(this->perimetrofig);

}

/**
 * @brief Calcula la superficie
 */
double Circulo::superficie(){
  this->area = ( PI*pow((this->radio),2) );
  return(this->area);
}

/**
 * @brief sobrecarga del operador ~ para circulo
 */
string Circulo::operator~(){
  string retorno;
  string strpuntos = "";
  for (int i=0; i < this->cantidadPuntos; i++){
      strpuntos = strpuntos + ~puntos[i];
      strpuntos = strpuntos + "\n";
  }
  retorno = strpuntos + "\n";
  retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
             "Mi color es "+this->color + " \n" +
             "Mi superficie es "+to_string(this->superficie())+ " \n" +
             "Mi perimetro es "+to_string(this->perimetro())+ " \n"
             "Mi radio es "+to_string(this->radio)+ " \n"
             );
  return retorno;
}

```

```
58 /**
59  * @brief destructor de la clase
60  */
61 Circulo::~Circulo(){};
```

4.7. Circulo.hpp

```
#ifndef CIRCULO_H
#define CIRCULO_H

    #include "./Includes.hpp"
    /**
     * @brief clase que se encarga e modelar un circulo
     */
    class Circulo : public Figura{
        public:
            Circulo(Vertice *puntos);
            ~Circulo();
            double perimetro();
            double superficie();
            string operator~();
            double radio;
    };

#endif
```

4.8. Rectangulo.cpp

```
#include "../include/Includes.hpp"

using namespace std;


/**
 * @brief contructor de la clase rectangulo
 * @param recibe un puntero a los vertices
 */
rectangulo::rectangulo(Vertice *temp){
    float distancias[3];
    float distancia_calc;
    this->puntos=temp;

    for (int i=0; i<3; i++){
        distancia_calc=puntos[0]>>puntos[i+1];
        distancias[i]=distancia_calc;
    }
    for (int i=0; i<3; i++){ //tres porque son los vertices restantes con los que debe comparar la distancia
        if (i==0){
            if (distancias[i] > distancias[i+1] && distancias[i] > distancias[i+2]){
                if (distancias[i+1] > distancias[i+2]){
                    this->base=distancias[i+1];
                    this->altura=distancias[i+2];
                }else{
                    this->base=distancias[i+2];
                    this->altura=distancias[i+1];
                }
            }
        }
        if (i==1){
            if (distancias[i] > distancias[i-1] && distancias[i] > distancias[i+1]){
                if (distancias[i-1] > distancias[i+1]){
                    this->base=distancias[i-1];
                    this->altura=distancias[i+1];
                }else{
                    this->base=distancias[i+1];
                    this->altura=distancias[i-1];
                }

            }
        }

        if (i==2){
            if (distancias[i] > distancias[i-2]  && distancias[i] > distancias[i-1]){
                 if (distancias[i-2] > distancias[i-1]){
                    this->base=distancias[i-2];
                    this->altura=distancias[i-1];
                }else{
                    this->base=distancias[i-1];
                    this->altura=distancias[i-2];
                }
            }
```

```
54
55          }
56
57      }
58      cout<< "La base es: " << this->base<< endl;
59      cout<< "La altura es: " << this->altura<< endl;
60      this->area = this->superficie();
61      this->perimetrofig = this->perimetro();
62      cout << "el area es: " << this->area << endl;
63      cout << "el perimetro es: " << this ->perimetrofig << endl;
64
65  }
66
67  /**
68   * @brief Calcula la superficie
69   */
70  double rectangulo::superficie(){
71    this->area=(this->altura)*(this->base);
72    return (this->area);
73
74  }
75
76  /**
77   * @brief Calcula el perimetro
78   */
79  double rectangulo::perimetro(){
80    this->perimetrofig=(2*(altura)+2*(this->base));
81    return (this->perimetrofig);
82
83  }
84
85  /**
86   * @brief destructor de la clase
87   */
88  rectangulo::~rectangulo(){}
89
90  /**
91   * @brief sobrecarga del operador ~ para circulo
92   */
93  string rectangulo::operator~(){
94    string retorno;
95    string strpuntos = "";
96    for (int i=0; i < this->cantidadPuntos; i++){
97        strpuntos = strpuntos + ~puntos[i];
98        strpuntos = strpuntos + "\n";
99    }
100   retorno = strpuntos + "\n";
101   retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
102             "Mi color es "+this->color + " \n" +
103             "Mi superficie es "+to_string(this->superficie())+ " \n" +
104             "Mi perimetro es "+to_string(this->perimetro())+ " \n"
105             "Mi base es "+to_string(this->base)+ " \n"
106             "Mi altura es "+to_string(this->altura)+ " \n"
107             );
108   return retorno;
109 }
```

4.9. Rectangulo.hpp

```
#ifndef RECTANGULO_H
#define RECTANGULO_H


    #include "./Includes.hpp"
    using namespace std;

    /**
     * @brief Clase rectangulo
     */
    class rectangulo : public Figura{
        public:
            float altura; //lo definiremos cuando tengamos el menu, pero los metodos ya depende de "este altura"
            float base; //lo definiremos cuando tengamos el menu, pero los metodos ya depende de "este base"
            rectangulo(Vertice *puntos);
            ~rectangulo();
            double superficie();
            double perimetro();
            string operator~();
    };
#endif
```

4.10. Triangulo.cpp

```
#include "../include/Includes.hpp"

using namespace std;


/**
 * @brief Constructor de tringulo
*/
triangulo::triangulo(){}

double triangulo::perimetro(){
  return ((this->lado1)+(this->lado2)+(this->lado3));
}
/**
 * @brief Destructor de tringulo
*/
triangulo::~triangulo(){}

/**
 * @brief sobrecarga del operador ~ para trinagulo
 */
string triangulo::operator~(){
  string retorno;
  string strpuntos = "";
  for (int i=0; i < this->cantidadPuntos; i++){
      strpuntos = strpuntos + ~puntos[i];
      strpuntos = strpuntos + "\n";
  }
  retorno = strpuntos + "\n";
  retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
             "Mi color es "+this->color + " \n" +
             "Mi superficie es "+to_string(this->superficie())+ " \n" +
             "Mi perimetro es "+to_string(this->perimetro())+ " \n"
             "Mi lado 1 es "+to_string(this->lado1)+ " \n"
             "Mi lado 2 es "+to_string(this->lado2)+ " \n"
             "Mi lado 3 es "+to_string(this->lado3)+ " \n"
             );
  return retorno;
}


//############################################################


//Especializaciones de triangulo

//No se a aden las foo perimetro, pues, eso est  en la superclase triangulo

//############################################################
/**
 * @brief Constructor de equilatero
 * @param temp es un puntero de un arreglo de vertices
 */
Equilatero::Equilatero(Vertice *temp){
  this->puntos=temp;
  this->lado1=puntos[0]>>puntos[1];
  this->lado2=puntos[1]>>puntos[2];
```

```
58    this->lado3=puntos[2]>>puntos[0];
59    cout << "Lado 1:" << this->lado1 << endl;
60    cout << "Lado 2:" << this->lado2 << endl;
61    cout << "Lado 3:" << this->lado3 << endl;
62    this->area = this->superficie();
63    this->perimetrofig = this->perimetro();
64    cout << "el area es: " << this->area << endl;
65    cout << "el perimetro es: " << this ->perimetrofig << endl;
66
67  }
68  /**
69   * @brief calcula la superficie
70   */
71  double Equilatero::superficie(){
72    return ((sqrt(3)/2) * pow(this->lado1,2));
73  }
74  Equilatero::~Equilatero(){}
75
76  /**
77   * @brief sobrecarga del operador ~ para equilatero
78   */
79  string Equilatero::operator~(){
80    string retorno;
81    string strpuntos = "";
82    for (int i=0; i < this->cantidadPuntos; i++){
83        strpuntos = strpuntos + ~puntos[i];
84        strpuntos = strpuntos + "\n";
85    }
86    retorno = strpuntos + "\n";
87    retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
88              "Mi color es "+this->color + " \n" +
89              "Mi superficie es "+to_string(this->superficie())+ " \n" +
90              "Mi perimetro es "+to_string(this->perimetro())+ " \n"
91              "Mi lado 1 es "+to_string(this->lado1)+ " \n"
92              "Mi lado 2 es "+to_string(this->lado2)+ " \n"
93              "Mi lado 3 es "+to_string(this->lado3)+ " \n"
94              "Soy un Equilatero" + " \n"
95              );
96    return retorno;
97  }
98  //#########################################################
99
100
101 //#########################################################
102 /**
103  * @brief Constructor de Escaleno
104  * @param temp es un puntero de un arreglo de vertices
105  */
106 Escaleno::Escaleno(Vertice *temp){
107   cout <<"Dennispurador: Constructor por defecto de Escaleno";
108   cout << endl;
109   this->puntos=temp;
110   this->lado1=puntos[0]>>puntos[1];
111   this->lado2=puntos[1]>>puntos[2];
112   this->lado3=puntos[2]>>puntos[0];
113   cout << "Lado 1:" << this->lado1 << endl;
114   cout << "Lado 2:" << this->lado2 << endl;
115   cout << "Lado 3:" << this->lado3 << endl;
116   this->area = this->superficie();
```

```
117   this->perimetrofig = this->perimetro();
118   cout << "el area es: " << this->area << endl;
119   cout << "el perimetro es: " << this ->perimetrofig << endl;
120 }
121
122 /**
123  * @brief destructor de Escaleno
124  */
125 Escaleno::~Escaleno(){
126     this->semiperimetro=(lado1+lado2+lado3)/2;
127 }
128 /**
129  * @brief calcula la superficie de un Escaleno
130  */
131 double Escaleno::superficie(){
132   return (sqrt(abs( semiperimetro * (s-this->lado1) * (s-this->lado2) * (s-this
      ->lado3))));
133 }
134
135 /**
136  * @brief sobrecarga del operador ~ para Escaleno
137  */
138 string Escaleno::operator~(){
139   string retorno;
140   string strpuntos = "";
141   for (int i=0; i < this->cantidadPuntos; i++){
142       strpuntos = strpuntos + ~puntos[i];
143       strpuntos = strpuntos + "\n";
144   }
145   retorno = strpuntos + "\n";
146   retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
147               "Mi color es "+this->color + " \n" +
148               "Mi superficie es "+to_string(this->superficie())+ " \n" +
149               "Mi perimetro es "+to_string(this->perimetro())+ " \n"
150               "Mi lado 1 es "+to_string(this->lado1)+ " \n"
151               "Mi lado 2 es "+to_string(this->lado2)+ " \n"
152               "Mi lado 3 es "+to_string(this->lado3)+ " \n"
153               "Soy un Escaleno" + " \n"
154               );
155   return retorno;
156 }
157
158 //#############################################################
159
160
161 //#############################################################
162
163 /**
164  * @brief Constructor de Isosceles
165  * @param temp es un puntero de un arreglo de vertices
166  */
167 Isosceles::Isosceles(Vertice *temp){
168     cout <<"Dennispurador: Constructor por defecto de Isosceles";
169     cout << endl;
170     this->puntos=temp;
171     float distancias[3];
172     distancias[0]=puntos[0]>>puntos[1];
173     distancias[1]=puntos[1]>>puntos[2];
174     distancias[2]=puntos[2]>>puntos[0];
```

```
175     for (int i=0; i<3; i++){ //tres porque son los vertices restantes con los
        que debe comparar la distancia
176         if (i==0){
177
178             if (distancias[i] > distancias[i+1] && distancias[i] > distancias[i
        +2]){
179                     this->lado1=distancias[i+1];
180                     this->lado2=distancias[i+2];
181                     this->lado3=distancias[i];
182             }
183         }
184         if (i==1){
185             if (distancias[i] > distancias[i-1] && distancias[i] > distancias[i
        +1]){
186                     this->lado1=distancias[i-1];
187                     this->lado2=distancias[i+1];
188                     this->lado3=distancias[i];
189             }
190         }
191
192         if (i==2){
193             if (distancias[i] > distancias[i-2]  && distancias[i] > distancias[
        i-1]){
194                     this->lado1=distancias[i-2];
195                     this->lado2=distancias[i-1];
196                     this->lado3=distancias[i];
197             }
198         }
199     }
200     cout << "Lado 1:" << this->lado1 << endl;
201     cout << "Lado 2:" << this->lado2 << endl;
202     cout << "Lado 3:" << this->lado3 << endl;
203     this->area = this->superficie();
204     this->perimetrofig = this->perimetro();
205     cout << "el area es: " << this->area << endl;
206     cout << "el perimetro es: " << this ->perimetrofig << endl;
207 }
208 /**
209  * @brief calcula la superficie
210  */
211 double Isosceles::superficie(){
212   return ( ((this->lado1)*(this->lado2))/2 );
213 }
214 Isosceles::~Isosceles(){}
215
216 /**
217  * @brief sobrecarga del operador ~ para Isosceles
218  */
219 string Isosceles::operator~(){
220   string retorno;
221   string strpuntos = "";
222   for (int i=0; i < this->cantidadPuntos; i++){
223       strpuntos = strpuntos + ~puntos[i];
224       strpuntos = strpuntos + "\n";
225   }
226   retorno = strpuntos + "\n";
227   retorno = retorno + ("Mi nombre es "+this->nombre + "  \n" +
228              "Mi color es "+this->color + " \n" +
229              "Mi superficie es "+to_string(this->superficie())+ " \n" +
```

```
230             "Mi perimetro es "+to_string(this->perimetro())+ " \n"
231             "Mi lado 1 es "+to_string(this->lado1)+ " \n"
232             "Mi lado 2 es "+to_string(this->lado2)+ " \n"
233             "Mi lado 3 es "+to_string(this->lado3)+ " \n"
234             "Soy un Isosceles" + " \n"
235             );
236   return retorno;
237 }
238 //#########################################################
```

4.11. Triangulo.hpp

```
#ifndef TRIANGULO_H
#define TRIANGULO_H

    #include "./Includes.hpp"

    using namespace std;


    /**
     * @brief calse tringulo
     */
    class triangulo : public Figura{
        public:
            float lado1;
            float lado2;
            float lado3;
            triangulo();
            ~triangulo();
            double perimetro();
            virtual string operator~();

    };

    /**
     * @brief calse Escaleno
     */

    class Escaleno : public triangulo{
        public:
            float semiperimetro; //Se completa en el menu (ecuacion=(lado1+
    lado2+lado3)/2)
            Escaleno(Vertice *puntos);
            ~Escaleno();
            double superficie();
            double s;
            string operator~();

    };

    /**
     * @brief calse equilatero
     */
    class Equilatero : public triangulo{
        public:
            Equilatero(Vertice *puntos);
            ~Equilatero();
            double superficie();
            string operator~();

    };



    /**
     * @brief calse Isosceles
     */
    class Isosceles : public triangulo{
```

```
57          public:
58              Isosceles(Vertice *puntos);
59              ~Isosceles();
60              double superficie();
61              string operator~();
62
63      };
64
65  #endif
```

4.12. Impresora.cpp

```
#include "../include/Includes.hpp"
using namespace std;

void nada (){
    cout << "madita" << endl;
}
```

4.13. Impresora.hpp

```
#ifndef IMPRESORA_H
#define IMPRESORA_H

using namespace std;


/**
 * @brief clase que imprmime objetos
 */
template <class T>
class Impresora {
    public:
        /**
         * @brief constructor por defecto
         */
        Impresora(){
        };
        /**
        * @brief constructor creado mediante template
        * @param fig es el objeto recibido por parametro
        */
        Impresora(T fig, string impresion){
            cout << ObtenerColor(fig.color);
            this->ruta  = fig.nombre;
            fig.impresion = impresion;
            imprimirPantalla(fig);
            imprimirArchivo(fig, fig.nombre);
            cout << FORMATO_ANSI_COLOR_RESET;
        };
        /**
         * @brief destructor por defecto de la clase
         */
        ~Impresora(){};

        /**
         * @brief clase que permite imprimir en el color correcto
         * @param nomColor es el color que debe buscar
         * @return debuelve el codigo de color que debe imprimir
         */
        string ObtenerColor(string nomColor){
            int tamanio = 6;
            string respuesta = "\x1b[01;37m";
            string nombre[tamanio] = {" Rojo"," Verde"," Amarillo"," Azul","
    Purpura"," Celeste"};
            string color[tamanio] = {"\x1b[31m","\x1b[32m","\x1b[33m","\x1b[34m
    ","\x1b[35m","\x1b[36m"};
            for (int i=0; i< tamanio ; i++ ){
                if (nomColor == nombre[i]){
                    respuesta = color[i];
                }
            }
            return respuesta;
        }

        /**
         * @brief metodo que permite imprmir la pantalla
         * @param objeto es el objeto que debe imprimir
```

```
56          */
57         void imprimirPantalla(const T &objeto){
58             cout << objeto.impresion << endl;
59         }
60         /**
61          * @brief metodo que permite imprmir a un archivo
62          * @param objeto es el objeto que debe imprimir
63          * @param sutaArchivo es la ruta del archivo a escribir
64          */
65         void imprimirArchivo(const T& objeto, string rutaArchivo){
66             ofstream archivo;
67             string ruta= "./"+ rutaArchivo;
68             archivo.open(ruta.c_str(),ios::out); //abriendo archivo
69             if (archivo.fail()){
70 <<<<<<< HEAD
71                 cout << "No se pudo abrir el archivo "  << this->nombre << endl
    ;
72                 exit(1);
73             }
74             archivo << ~lin;
75             cout << "se creo el archivo " << this->nombre << endl;
76 =======
77                 cout << "No se pudo abrir el archivo "  << rutaArchivo << endl;
78                 exit(1);
79             }
80             archivo << objeto.impresion;
81             cout << "se creo el archivo " << rutaArchivo << endl;
82 >>>>>>> ead433545a5c7623452a7be129b473abd71f7fb0
83             archivo.close();
84         }
85         string ruta;
86 };
87
88 #endif
```

4.14. main.cpp

```
/**
 * @file main.c
 * @author Jesus Zu iga Mendez
 * @author Dennis Chavarria Soto
 * @brief Archivo pricipal, Laboratorio sobre herencia representada con
    geometria
 * @version 1.0
 * @date 24 de setiembre de 2019
 * @copyright Copyleft (l) 2019
 */

#include "./include/Includes.hpp"

using namespace std;


/**
 * @brief Funcion main del codigo
 */
int main(int argc, char** argv){
    Principal clasePrincipal;
    clasePrincipal.Bienvenido();
    int opcion = 0;
    do{
        clasePrincipal.Menu();
        cin >> opcion;
        switch (opcion)
        {
        case 1:
            clasePrincipal.MtdRectangulo();
            break;
        case 2:
            clasePrincipal.MtdTriangulo();
            break;
        case 3:
            clasePrincipal.MtdCirculo();
            break;
        case 4:
            clasePrincipal.MtdCualquierFigura();
            break;
        default:
            break;
        }
    }while (opcion != 0);

    return 0;
}
```

4.15. Includes.hpp

```
#ifndef INCLUDES_H
#define INCLUDES_H

    #define PI 3.1415
    #include <iostream>
    #include <math.h>
```

```
7     #include <fstream>
8     #include "./Formato.hpp"
9     #include "./Vertice.hpp"
10    #include "./Figura.hpp"
11    #include "./Impresora.hpp"
12    #include "./Triangulo.hpp"
13    #include "./Rectangulo.hpp"
14    #include "./Circulo.hpp"
15    #include "./Impresora.hpp"
16    #include "./Tools.hpp"
17
18 #endif
```

4.16. Tools.hpp

```
#ifndef TOOLS_H
#define TOOLS_H
    #include "./Includes.hpp"

    class Principal{
        public:
            Principal(){};
            ~Principal(){};
            /**
             * @brief metodo que imprime un banner de bienvenida
             */
            void Bienvenido(){
                cout << FORMATO_ANSI_COLOR_LIGHT_BLUE << endl;
                cout << "  _____   _                                _     _       " <<
    endl;
                cout << "| __  ||_| ___  ___  _ _  ___  ___ |_| _| | ___ " <<
    endl;
                cout << "| __ -|| || -_||   || | || -_||   || || . || . |" <<
    endl;
                cout << "|_____||_||___||_|_| \\_/ |___||_|_||_||___||___|" <<
    endl;
                cout << FORMATO_ANSI_COLOR_RESET << endl;
            }
            /**
             * @brief metodo que imprime un un menu
             */
            void Menu(){
                cout << "Digite la opcion que desea realizar" << endl;
                cout << "   1: Ingresar un .....Rectangulo" << endl;
                cout << "   2: Ingrear un ......Triangulo" << endl;
                cout << "   3: Ingresar un .....Circulo" << endl;
                cout << "   4: Ingresar una figura distinta con N cantidad de
    lados "<<FORMATO_BLINK_EFFECT<<"(EN Construccion.....)" << endl;
                cout << "   5: Manipular figuras"<<FORMATO_BLINK_EFFECT<<"(EN
    Construccion.....)" << endl;
                cout << "   0: Salir" << endl;
            }

            /**
             * @brief metodo que crea un arreglo de objetos tipo vertice
             * @param cantidad es el numero de vertices
             * @param arreglo es el puntero del arreglo que se va a pasar
             */
            void HacerArregloVertices(int cantidad, Vertice* arreglo){
                int x = 0;
                int y = 0;
                cout << "Digite los vertices en orden siguiendo las manecillas
    del reloj" << endl;
                for (int i = 0; i< cantidad; i++){
                    cout << "Vertice # " << (i+1) << endl;
                    cout << "Digite la coordenada en X" << endl;
                    cin >> x;
                    cout << "Digite la coordenada en Y" << endl;
                    cin >> y;
                    arreglo[i].x = x;
                    arreglo[i].y = y;
                    arreglo[i].identificador = i;
```

```
51                  }
52              }
53              /**
54               * @brief metodo que permite escoger un color de figura
55               */
56              string Color(){
57                  int tamanio = 6;
58                  string respuesta = "Blanco";
59                  string nombre[tamanio] = {" Rojo"," Verde"," Amarillo"," Azul",
    " Purpura"," Celeste"};
60                  string color[tamanio] = {"\x1b[31m","\x1b[32m","\x1b[33m","\x1b
    [34m","\x1b[35m","\x1b[36m"};
61
62                  cout << "Seleccione el color de la figura(Blanco por defecto)"
    << endl;
63                  for (int i =0; i < tamanio; i++){
64                      cout << "\x1b[01;37m";
65                      cout << "       " << (i+1) << ": " << color[i] << nombre[i]
     << endl;
66                  }
67                  int seleccion = 0;
68                  cout << FORMATO_ANSI_COLOR_RESET;
69                  cin >> seleccion;
70                  if (seleccion <= tamanio){
71                      respuesta = nombre[seleccion-1];
72                  }
73                  return respuesta;
74              }
75              /**
76               * @brief metodo que instancia un objeto de tipo rectangulo
77               */
78              void MtdRectangulo(){
79                  Vertice arregloVertices[4];
80                  HacerArregloVertices(4, arregloVertices);
81                  rectangulo figura(arregloVertices);
82                  figura.cantidadPuntos = 4;
83                  string lectura = "";
84                  cout << "Digite el nombre de la figura" << endl;
85                  cin >> lectura;
86                  figura.nombre = lectura;
87                  figura.color = Color();
88                  figura.impresion = ~figura;
89                  Impresora<rectangulo> imprimir (figura , figura.impresion);
90              }
91              /**
92               * @brief metodo que instancia un objeto de tipo Circulo
93               */
94              void MtdCirculo(){
95                  Vertice arregloVertices[2];
96                  HacerArregloVertices(2, arregloVertices);
97                  Circulo figura (arregloVertices);
98                  figura.cantidadPuntos = 2;
99                  string lectura = "";
100                 cout << "Digite el nombre de la figura" << endl;
101                 cin >> lectura;
102                 figura.nombre = lectura;
103                 figura.color = Color();
104                 figura.impresion = ~figura;
105                 Impresora<Circulo> imprimir (figura, figura.impresion);
```

```
106             }
107             /**
108              * @brief metodo que instancia un objeto de tipo Triangulo
109              */
110             void MtdTriangulo(){
111                 Vertice arregloVertices[3];
112                 HacerArregloVertices(3, arregloVertices);
113                 float distancias[3];
114                 distancias[0]=arregloVertices[0]>>arregloVertices[1];
115                 distancias[1]=arregloVertices[1]>>arregloVertices[2];
116                 distancias[2]=arregloVertices[2]>>arregloVertices[0];
117                 if ((distancias[0] == distancias[1]) && (distancias[0] ==
    distancias[2])){
118                     Equilatero figura(arregloVertices);
119                     figura.cantidadPuntos = 3;
120                     string lectura = "";
121                     cout << "Digite el nombre de la figura" << endl;
122                     cin >> lectura;
123                     figura.nombre = lectura;
124                     figura.color = Color();
125                     figura.impresion = ~figura;
126                     Impresora<Equilatero> imprimir (figura, figura.impresion);
127                 }else if ((distancias[0] == distancias[1]) || (distancias[0] ==
     distancias[2]) || (distancias[1] == distancias[2])){
128                     Isosceles figura(arregloVertices);
129                     figura.cantidadPuntos = 3;
130                     string lectura = "";
131                     cout << "Digite el nombre de la figura" << endl;
132                     cin >> lectura;
133                     figura.nombre = lectura;
134                     figura.color = Color();
135                     figura.impresion = ~figura;
136                     Impresora<Isosceles> imprimir (figura, figura.impresion);
137                 }else{
138                     Escaleno figura(arregloVertices);
139                     figura.cantidadPuntos = 3;
140                     string lectura = "";
141                     cout << "Digite el nombre de la figura" << endl;
142                     cin >> lectura;
143                     figura.nombre = lectura;
144                     figura.color = Color();
145                     figura.impresion = ~figura;
146                     Impresora<Escaleno> imprimir (figura, figura.impresion);
147                 }
148             }
149             /**
150              * @brief metodo que instancia un objeto n cantidad de ladoso
151              */
152             void MtdCualquierFigura(){
153                 // int cantidad = 0;
154                 // cout << "Digite la cantidad de vertices" << endl;
155                 // cin >> cantidad;
156                 // Vertice arregloVertices[cantidad];
157                 // HacerArregloVertices(cantidad, arregloVertices);
158                 // Figura figura = new Circulo(arregloVertices);
159                 // figura.cantidadPuntos = cantidad;
160                 // string lectura = "";
161                 // cout << "Digite el nombre de la figura" << endl;
162                 // cin >> lectura;
```

```
163                 // figura.nombre = lectura;
164                 // figura.color = Color();
165                 // figura.impresion = ~figura;
166                 // Impresora<Escaleno> imprimir (figura, figura.impresion);
167             }
168     };
169 #endif
```