
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

# EJERCICIOS DE CLASE Nº 2

**NOMBRE COMPLETO:** Uriarte Ortiz Enrique Yahir

**Nº de Cuenta:** 318234757

**GRUPO DE LABORATORIO:** 02

**GRUPO DE TEORÍA:** 04

**SEMESTRE 2025-1**

**FECHA DE ENTREGA LÍMITE:** Martes 20 de Agosto del 2024

**CALIFICACIÓN: _____**

## EJERCICIOS DE SESIÓN:

1. Actividades realizadas.

*1)* Generar las figuras copiando los vértices de *"triangulorojo"* y *"cuadradoverde"*:
Para el desarrollo de esta actividad únicamente copié los códigos de los *"GLfloat"* para el triángulo y el cuadrado y cambié los colores de cada de acuerdo con lo solicitado. Para poder visualizar la creación de cada figura únicamente se cambia el valor de *"meshColorList"* ubicado en la función main, en cada ejecución del codigo.

Capturas de pantalla del código:

- Triángulo azul.

```
GLfloat vertices_trianguloazul[] = {//Triángulo  azul, meshColorList[1] -- DE ACTIVIDAD 1
    //    X      Y     Z      R   G    B
        -1.0f, -1.0f, 0.5f, 0.0f,0.0f,1.0f,
         1.0f, -1.0f, 0.5f, 0.0f,0.0f,1.0f,
         0.0f,  1.0f, 0.5f, 0.0f,0.0f,1.0f };
MeshColor* trianguloazul = new MeshColor();
trianguloazul->CreateMeshColor(vertices_trianguloazul, 18);
meshColorList.push_back(trianguloazul);
```

Fig. 1. Código de la figura.

```
//Posicion DE FIGURAS
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();
```

Fig. 2. Código para seleccionar la figura a mostrar en pantalla.

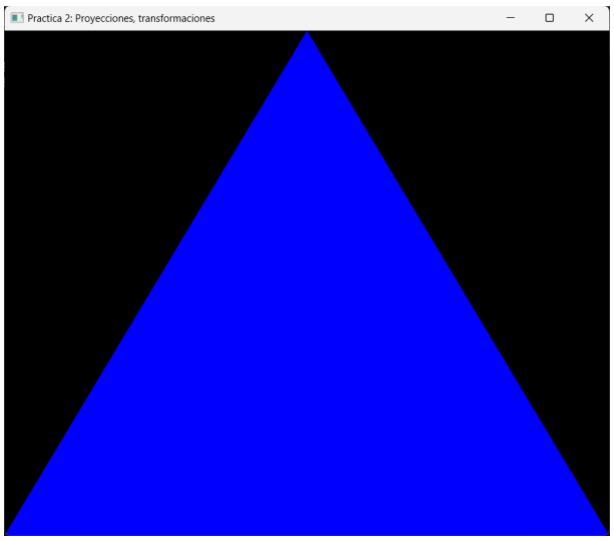


Fig. 3. Ejecución de la figura.

- Triangulo verde (0,0.5,0)

```
GLfloat vertices_trianguloverde[] = {//Triangulo verde, meshColorList[2] -- DE ACTIVIDAD 1
    //    X      Y     Z      R   G    B
        -1.0f, -1.0f, 0.5f, 0.0f,0.5f,0.0f,
         1.0f, -1.0f, 0.5f, 0.0f,0.5f,0.0f,
         0.0f,  1.0f, 0.5f, 0.0f,0.5f,0.0f };
MeshColor* trianguloverde = new MeshColor();
trianguloverde->CreateMeshColor(vertices_trianguloverde, 18);
meshColorList.push_back(trianguloverde);
```

Fig. 4. Código de figura.

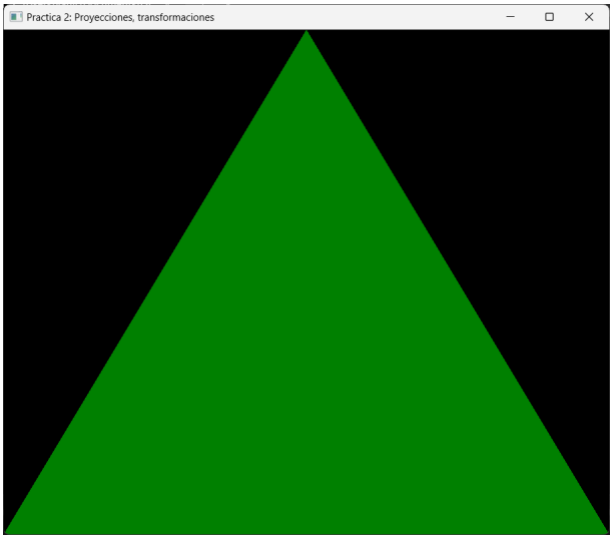


Fig. 5. Ejecución de la figura.

- Cuadrado rojo.

```
GLfloat vertices_cuadradorojo[] = {//Cuadrado rojo, meshColorList[3] -- DE ACTIVIDAD 1
//    X      Y     Z     R    G    B
    -0.5f, -0.5f, 0.5f, 1.0f,0.0f,0.0f,
     0.5f, -0.5f, 0.5f, 1.0f,0.0f,0.0f,
     0.5f,  0.5f, 0.5f, 1.0f,0.0f,0.0f,
    -0.5f, -0.5f, 0.5f, 1.0f,0.0f,0.0f,
     0.5f,  0.5f, 0.5f, 1.0f,0.0f,0.0f,
    -0.5f,  0.5f, 0.5f, 1.0f,0.0f,0.0f };
MeshColor* cuadradorojo = new MeshColor();
cuadradorojo->CreateMeshColor(vertices_cuadradorojo, 36);
meshColorList.push_back(cuadradorojo);
```

Fig. 6. Código de figura.

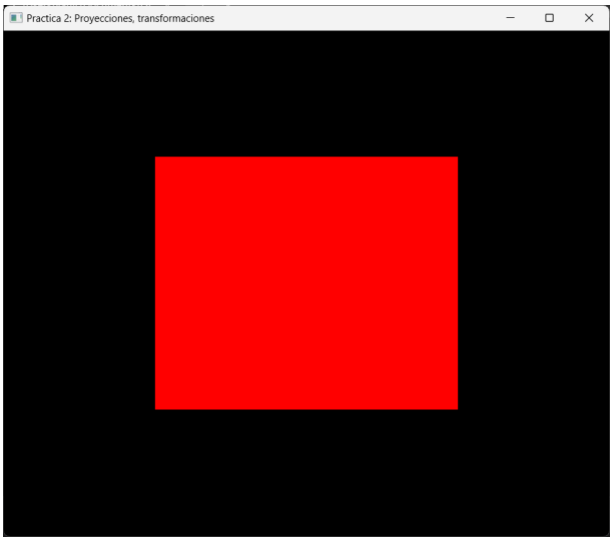


Fig. 7. Ejecución de la figura.

- Cuadrado verde.

```
GLfloat vertices_cuadradoverde[] = {//Cuadrado verde, meshColorList[4] -- DE ACTIVIDAD 1
    //    X      Y     Z     R    G    B
        -0.5f, -0.5f, 0.0f, 0.0f,1.0f,0.0f,
         0.5f, -0.5f, 0.0f, 0.0f,1.0f,0.0f,
         0.5f,  0.5f, 0.0f, 0.0f,1.0f,0.0f,
        -0.5f, -0.5f, 0.0f, 0.0f,1.0f,0.0f,
         0.5f,  0.5f, 0.0f, 0.0f,1.0f,0.0f,
        -0.5f,  0.5f, 0.0f, 0.0f,1.0f,0.0f };
MeshColor* cuadradoverde = new MeshColor();
cuadradoverde->CreateMeshColor(vertices_cuadradoverde, 36);
meshColorList.push_back(cuadradoverde);
```

Fig. 8. Código de figura.

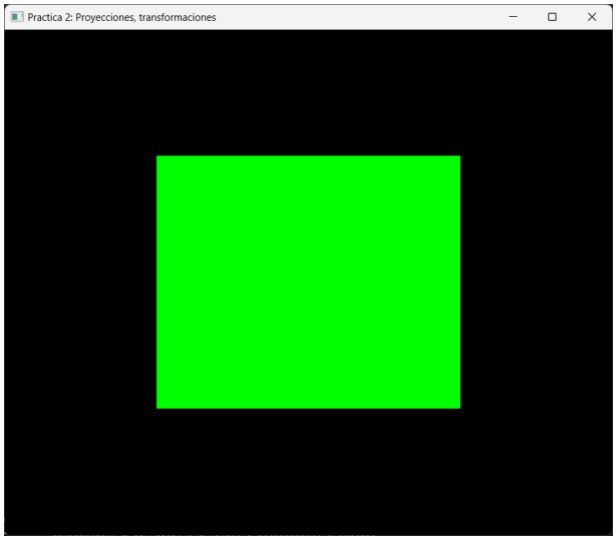


Fig. 9. Ejecución de la figura.

- Cuadrado café (0.478, 0.255, 0.067)

```
GLfloat vertices_cuadradocafe[] = {//Cuadrado cafe, meshColorList[5] -- DE ACTIVIDAD 1
    //    X      Y     Z       R       G     B
        -0.5f, -0.5f, 0.0f, 0.478f, 0.255f,0.067f,
         0.5f, -0.5f, 0.0f, 0.478f, 0.255f,0.067f,
         0.5f,  0.5f, 0.0f, 0.478f, 0.255f,0.067f,
        -0.5f, -0.5f, 0.0f, 0.478f, 0.255f,0.067f,
         0.5f,  0.5f, 0.0f, 0.478f, 0.255f,0.067f,
        -0.5f,  0.5f, 0.0f, 0.478f, 0.255f,0.067f };
MeshColor* cuadradocafe = new MeshColor();
cuadradocafe->CreateMeshColor(vertices_cuadradocafe, 36);
meshColorList.push_back(cuadradocafe);
```

Fig. 10. Código de figura.

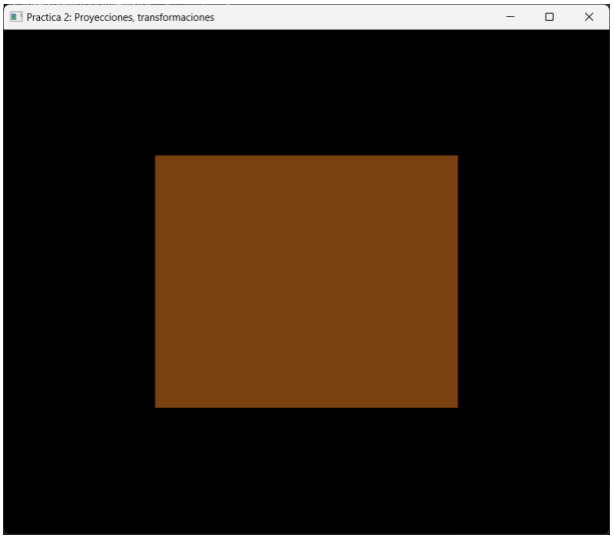


Fig. 11. Ejecución de la figura.

*2)* Usando la proyección ortogonal generar el siguiente dibujo a partir de instancias de las figuras anteriormente creadas, recordar que todos se dibujan en el origen y por transformaciones geométricas se desplazan.
Para esta segunda actividad tuve que modificar las dimensiones de las figuras ya que no se parecían a las que tenían las figuras en la imagen de muestra para la actividad, además realicé un dibujo boceto en una hoja cuadriculada para tener mejor referencia de las dimensiones que debían tener cada figura, pero aún se mantenían definidas en el origen.

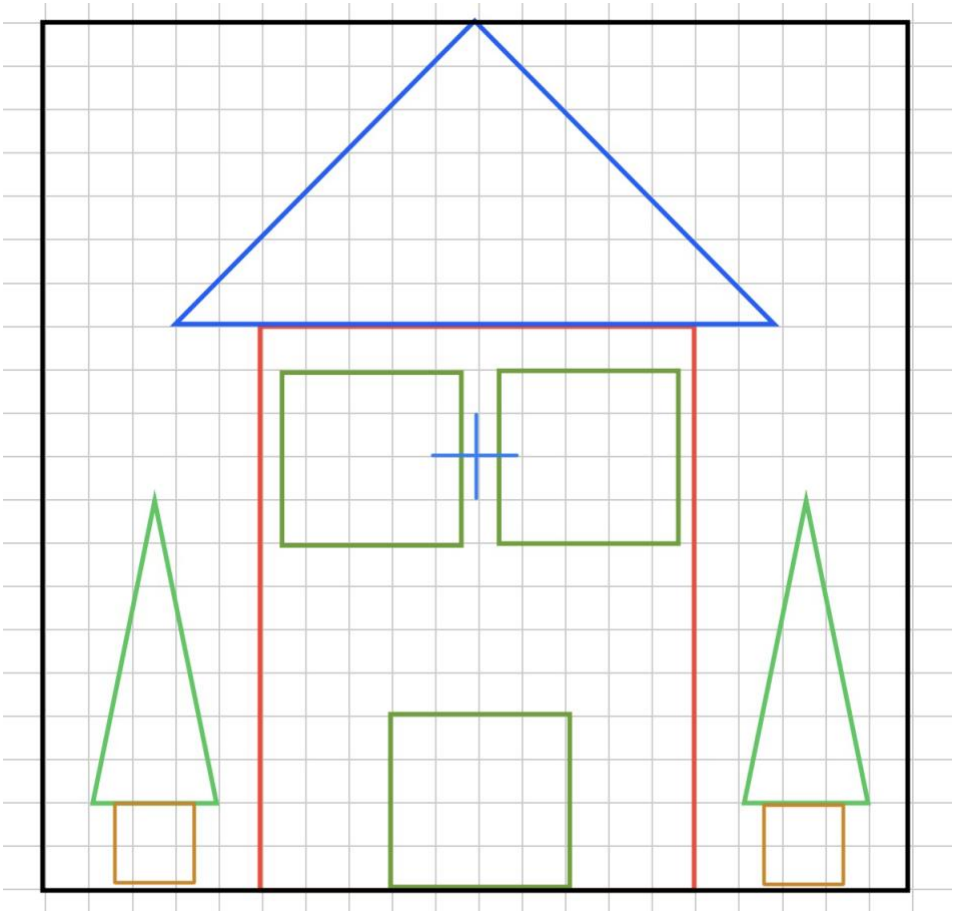

Fig. 12. Boceto para las coordenadas de las figuras.

Capturas de pantalla del código:

```
GLfloat vertices_trianguloazul[] = {//Triángulo  azul, meshColorList[1] -- DE ACTIVIDAD 2
//    X      Y     Z      R   G    B
    -0.6f, -0.35f, 0.0f,    0.0f,0.0f,1.0f,
     0.6f, -0.35f, 0.0f,    0.0f,0.0f,1.0f,
     0.0f,  0.35f, 0.0f,    0.0f,0.0f,1.0f };
MeshColor* trianguloazul = new MeshColor();
trianguloazul->CreateMeshColor(vertices_trianguloazul, 18);
meshColorList.push_back(trianguloazul);
```

Fig. 13. Código de triángulo azul ajustado.

```
GLfloat vertices_trianguloverde[] = {//Triangulo verde, meshColorList[2] -- DE ACTIVIDAD 2
//    X      Y     Z      R   G    B
    -0.15f, -0.35f, 0.0f, 0.0f,0.5f,0.0f,
     0.15f, -0.35f, 0.0f, 0.0f,0.5f,0.0f,
     0.0f,   0.35f, 0.0f, 0.0f,0.5f,0.0f };
MeshColor* trianguloverde = new MeshColor();
trianguloverde->CreateMeshColor(vertices_trianguloverde, 18);
meshColorList.push_back(trianguloverde);
```

Fig. 14. Código de triángulo verde ajustado.

Para el cuadrado rojo tuve que modificarlo en tamaño de largo para que se pareciera más a la imagen de referencia ya que en esta más bien era un rectángulo.

```
GLfloat vertices_cuadradorojo[] = {//Rectangulo rojo, meshColorList[3] -- DE ACTIVIDAD 2
    //    X      Y     Z     R    G    B
        -0.5f, -0.75f, 0.0f,    1.0f,0.0f,0.0f,
         0.5f, -0.75f, 0.0f,    1.0f,0.0f,0.0f,
         0.5f,  0.75f, 0.0f,    1.0f,0.0f,0.0f,
        -0.5f, -0.75f, 0.0f,    1.0f,0.0f,0.0f,
         0.5f,  0.75f, 0.0f,    1.0f,0.0f,0.0f,
        -0.5f,  0.75f, 0.0f,    1.0f,0.0f,0.0f };
MeshColor* cuadradorojo = new MeshColor();
cuadradorojo->CreateMeshColor(vertices_cuadradorojo, 36);
meshColorList.push_back(cuadradorojo);
```

Fig. 15. Código de rectángulo rojo ajustado.

```
GLfloat vertices_cuadradoverde[] = {//Cuadrado verde, meshColorList[4] -- DE ACTIVIDAD 2
    //    X      Y     Z     R    G    B
        -0.2f, -0.2f, 0.0f, 0.0f,1.0f,0.0f,
         0.2f, -0.2f, 0.0f, 0.0f,1.0f,0.0f,
         0.2f,  0.2f, 0.0f, 0.0f,1.0f,0.0f,
        -0.2f, -0.2f, 0.0f, 0.0f,1.0f,0.0f,
         0.2f,  0.2f, 0.0f, 0.0f,1.0f,0.0f,
        -0.2f,  0.2f, 0.0f, 0.0f,1.0f,0.0f };
MeshColor* cuadradoverde = new MeshColor();
cuadradoverde->CreateMeshColor(vertices_cuadradoverde, 36);
meshColorList.push_back(cuadradoverde);
```

Fig. 16. Código de cuadrado verde ajustado.

```
GLfloat vertices_cuadradocafe[] = {//Cuadrado cafe, meshColorList[5] -- DE ACTIVIDAD 2
    //    X      Y     Z       R       G     B
        -0.1f, -0.1f, 0.0f, 0.478f, 0.255f,0.067f,
         0.1f, -0.1f, 0.0f, 0.478f, 0.255f,0.067f,
         0.1f,  0.1f, 0.0f, 0.478f, 0.255f,0.067f,
        -0.1f, -0.1f, 0.0f, 0.478f, 0.255f,0.067f,
         0.1f,  0.1f, 0.0f, 0.478f, 0.255f,0.067f,
        -0.1f,  0.1f, 0.0f, 0.478f, 0.255f,0.067f };
MeshColor* cuadradocafe = new MeshColor();
cuadradocafe->CreateMeshColor(vertices_cuadradocafe, 36);
meshColorList.push_back(cuadradocafe);
```

Fig. 17. Código de cuadrado cafe ajustado.

También modifiqué la parte en la función main que define las dimensiones de la pantalla para que de igual manera se pareciera más a la imagen de la actividad.

```
int main(){
    mainWindow = Window(900, 750);
    mainWindow.Initialise();
    CreaPiramide();         //índice 0 en MeshList
    CrearCubo();            //índice 1 en MeshList
    CrearLetrasyFiguras();  //usa MeshColor, índices en MeshColorList
    CreateShaders();
    GLuint uniformProjection = 0;
    GLuint uniformModel = 0;
```

Fig. 18. Código ajuste en dimensiones de pantalla.

En la proyección ortogonal modifiqué la distancia de cercanía, esto lo hice en un inicio por si colocaba la distancia de en el eje Z mayor a 0.1 al momento de desplazar las figuras.

```
glm::mat4 projection = glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 100.0f);
```

Fig. 19. Código ajuste en proyección ortogonal.

Después establecí las coordenadas para cada figura de acuerdo con la imagen de referencia, para asegurarme de que los cuadrados verdes se colocaran enfrente del rectángulo rojo, coloqué el rectángulo rojo una posición más abajo en el eje Z que la de los demás.

```
//Posicion cuadrado rojo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, -5.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();

//Posicion puerta
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.8f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();

//Posicion Ventana 1
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, -0.1f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();
```

Fig. 20. Código desplazamiento de figuras.

```
//Posicion Ventana 2
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.25f, -0.1f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();

//Posicion Techo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.6f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();
```

Fig. 21. Código desplazamiento de figuras.

```
//Posicion Tronco 1
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.9f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[5]->RenderMeshColor();

//Posicion Tronco 2
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.9f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[5]->RenderMeshColor();

//Posicion arbol 1
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.45f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();

//Posicion arbol 2
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.45f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();
```

Fig. 23. Código desplazamiento de figuras.


Fig. 24. Ejecución final del código.

2. *Problemas presentados.*

   Para la primera actividad no tuve ninguna complicación o error, y en la segunda actividad lo más difícil fue recordar el funcionamiento del desplazamiento con el eje Z de las figuras, lo más tardado fue encontrar la ubicación de desplazamiento correcta para las figuras y al mismo tiempo las dimensiones correctas a las que debían estar las figuras para que se pareciera más a la imagen de referencia

3. *Conclusión:*

   Después de realizar los ejercicios propuestos, considero que la complejidad que tenían era normal y de hecho siento que los ejercicios fueron más sencillos que los de la práctica anterior, pero aún así sirvieron para práctica la forma de ubicar los elementos en la pantalla de ejecución. Considero que los ejercicios de la practica estuvieron al nivel de lo enseñado