MANUAL TECNICO

PROYECTO PROGRAMACION 1

6590 – 18 – 11575

EMERSON FRANCISCO GARCIA VASQUEZ

Introducción

El programa de graficación de figuras geométricas es una aplicación desarrollada en C++ utilizando el IDE CodeBlocks. Está diseñado para permitir al usuario dibujar diversas figuras geométricas en una pantalla circular, con la capacidad de personalizar el carácter utilizado para dibujar y el color del mismo. Este manual proporciona información detallada sobre la estructura del programa, sus características, requisitos y cómo compilarlo.

Universidad Mariano Gálvez de Guatemala
PROGRAMACION 1

Requisitos del Sistema

Sistema Operativo: Windows

IDE: CodeBlocks (o cualquier otro IDE de preferencia)

Compilador: MinGW (incluido con CodeBlocks)

Instalación y Configuración

Descargue e instale CodeBlocks desde el sitio web oficial.

Asegúrese de tener el compilador MinGW configurado correctamente con CodeBlocks.

Descargue el código fuente del programa de graficación de figuras geométricas.

Abra el proyecto en CodeBlocks y compile el código.

Estructura del Código

El código del programa se divide en varias secciones principales:

Funciones de Dibujo de Figuras Geométricas:

Cada función está diseñada para dibujar una figura geométrica específica, como triángulos, cuadrados, rectángulos, círculos, líneas, rombos y hexágonos.

Utiliza funciones de la API de Windows para manipular la posición del cursor en la pantalla y dibujar los puntos que representan la figura.

Interacción con el Usuario:

El programa proporciona una interfaz simple para que el usuario pueda interactuar.

Permite al usuario mover un cursor en la pantalla con las teclas de flecha.

Al presionar la tecla F12, se muestra un menú con opciones para dibujar diferentes figuras geométricas.

Manejo de Eventos de Teclado:

Utiliza las funciones _kbhit() y _getch() para detectar y leer las teclas presionadas por el usuario.

Ejecuta el código correspondiente para manejar el evento cuando se presiona una tecla.

Dibujo en Pantalla Circular:

Implementa funciones trigonométricas para calcular las coordenadas de los puntos en la circunferencia de la pantalla.

Asegura que las figuras se dibujen correctamente incluso cuando parte de ellas esté cerca de los bordes de la pantalla circular.

Ciclo Principal del Programa:

El bucle principal del programa se ejecuta continuamente hasta que el usuario decide salir del programa.

Se verifica si se ha presionado alguna tecla y se ejecuta el código correspondiente para manejar el evento.

Funcionamiento del Programa

Al ejecutar el programa, el usuario puede mover un cursor en la pantalla utilizando las teclas de flecha.

Presionando la tecla F12, se muestra un menú con opciones para dibujar diferentes figuras geométricas.

El usuario puede seleccionar la figura deseada y proporcionar los parámetros necesarios para dibujarla.

El programa utiliza las funciones de dibujo correspondientes para graficar la figura en la posición actual del cursor.

Código del programa

```
#include <iostream>

#include <conio.h>

#include <windows.h>

#include <cmath>


using namespace std;


void setCursorPosition(int x, int y) {

   COORD coord;
```


```
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

void clearScreen() {
    system("cls");
}

bool isKeyPressed(int key) {
    return GetAsyncKeyState(key) & 0x8000;
}

void drawHexagon(int sideLength, int x, int y) {
    int startX = x;
    int startY = y;

    for (int i = 0; i < 6; ++i) {
        int nextX = startX + sideLength * cos((i * 60) * 3.14159 / 180);
        int nextY = startY + sideLength * sin((i * 60) * 3.14159 / 180);
        int nextX2 = startX + sideLength * cos(((i + 1) * 60) * 3.14159 / 180);
        int nextY2 = startY + sideLength * sin(((i + 1) * 60) * 3.14159 / 180);
        setCursorPosition(nextX, nextY);
        cout << ".";
        setCursorPosition(nextX2, nextY2);
        cout << ".";
    }
}
```


```
void drawTriangle(int baseLength, int height, int x, int y) {
    int startX = x - baseLength / 2;
    int startY = y - height / 2;
    int currentX, currentY;

    currentY = startY;
    for (int i = 0; i < height; ++i) {
        currentX = startX + i;
        setCursorPosition(currentX, currentY);
        cout << ".";
        setCursorPosition(startX + height - 1, currentY);
        cout << ".";
        --currentY;
    }
    currentX = startX;
    currentY = startY;
    for (int i = 0; i < baseLength; ++i) {
        setCursorPosition(currentX, currentY);
        cout << ".";
        ++currentX;
    }
}

void drawSquare(int sideLength, int x, int y) {
    int startX = x - sideLength / 2;
    int startY = y - sideLength / 2;

    for (int i = 0; i < sideLength; ++i) {
```


```
        setCursorPosition(startX + i, startY);
        cout << ".";
        setCursorPosition(startX + i, startY + sideLength - 1);
        cout << ".";
    }
    for (int i = 1; i < sideLength - 1; ++i) {
        setCursorPosition(startX, startY + i);
        cout << ".";
        setCursorPosition(startX + sideLength - 1, startY + i);
        cout << ".";
    }
}

void drawRectangle(int width, int height, int x, int y) {
    int startX = x - width / 2;
    int startY = y - height / 2;

    for (int i = 0; i < width; ++i) {
        setCursorPosition(startX + i, startY);
        cout << ".";
        setCursorPosition(startX + i, startY + height - 1);
        cout << ".";
    }
    for (int i = 1; i < height - 1; ++i) {
        setCursorPosition(startX, startY + i);
        cout << ".";
        setCursorPosition(startX + width - 1, startY + i);
        cout << ".";
    }
```


```
}

void drawCircle(int radius, int x, int y) {
    int startX = x - radius;
    int startY = y - radius;

    for (int i = -radius; i <= radius; ++i) {
        for (int j = -radius; j <= radius; ++j) {
            if (abs(i * i + j * j - radius * radius) < radius) {
                setCursorPosition(startX + j, startY + i);
                cout << ".";
            }
        }
    }
}

void drawLine(int length, const string& orientation, int x, int y) {
    int startX = x;
    int startY = y;

    if (orientation == "horizontal") {
        for (int i = 0; i < length; ++i) {
            setCursorPosition(startX + i, startY);
            cout << ".";
        }
    } else if (orientation == "vertical") {
        for (int i = 0; i < length; ++i) {
            setCursorPosition(startX, startY + i);
            cout << ".";
```

```
    }
  } else if (orientation == "diagonal") {
    for (int i = 0; i < length; ++i) {
      setCursorPosition(startX + i, startY + i);
      cout << ".";
    }
  }
}

void drawDiamond(int sideLength, int x, int y) {
  int startX = x;
  int startY = y;

  for (int i = 0; i <= sideLength; ++i) {
    setCursorPosition(startX - i, startY);
    cout << ".";
    setCursorPosition(startX + i, startY);
    cout << ".";
  }
  for (int i = sideLength - 1; i >= 0; --i) {
    setCursorPosition(startX - i, startY);
    cout << ".";
    setCursorPosition(startX + i, startY);
    cout << ".";
    --startY; // Moving one row up for each iteration
  }
}
```


```
int main() {
    int screenWidth = 80;
    int screenHeight = 25;

    int x = screenWidth / 2;
    int y = screenHeight / 2;
    char keyPressed;

    do {
        clearScreen();

        if (!isKeyPressed(VK_F12)) {
            setCursorPosition(0, 0);
            cout << "Presione las flechas para mover el cursor." << endl << endl;
            cout << "Presione F12 para ver el menu." << endl << endl;
            cout << "Presione Esc 2 veces para cerrar el programa." << endl;
            cout << endl;

            setCursorPosition(x, y);
            cout << ".";
        }

        if (_kbhit()) {
            keyPressed = _getch();
            switch (keyPressed) {
                case 72: // Flecha arriba
                    if (y > 0 && !isKeyPressed(VK_F12))
                        y -= 1;
                    break;
```


```
        case 80: // Flecha abajo
            if (y < screenHeight - 1 && !isKeyPressed(VK_F12))
                y += 1;
            break;
        case 75: // Flecha izquierda
            if (x > 0 && !isKeyPressed(VK_F12))
                x -= 1;
            break;
        case 77: // Flecha derecha
            if (x < screenWidth - 1 && !isKeyPressed(VK_F12))
                x += 1;
            break;
        case 27: // Tecla Esc
            if (!isKeyPressed(VK_F12)) {
                x = screenWidth / 2;
                y = screenHeight / 2;
            }
            break;
        case 120: // Tecla F9
            clearScreen();
            break;
    }
}

if (isKeyPressed(VK_F12)) {
    clearScreen();
    setCursorPosition(25, 0);
    cout << "MENU " << endl;
    setCursorPosition(0, 2);
```

```
cout << "F1: Triangulo " << endl;
setCursorPosition(15, 2);
cout << "F2: Cuadrado" << endl;
setCursorPosition(35, 2);
cout << "F3: Rectangulo" << endl;
setCursorPosition(55, 2);
cout << "F4: Circulo" << endl;
setCursorPosition(70, 2);
cout << "F5: Linea" << endl;
setCursorPosition(0, 3);
cout << "F6: Rombo" << endl;
setCursorPosition(15, 3);
cout << "F7: Hexagono" << endl;

int menuX = 0;
int menuY = 4;

do {
   if (_kbhit()) {
      keyPressed = _getch();
      switch (keyPressed) {
         case 72: // Flecha arriba
            if (menuY > 0)
               menuY -= 1;
            break;
         case 80: // Flecha abajo
            if (menuY < screenHeight - 1)
               menuY += 1;
            break;
```


```
        case 75: // Flecha izquierda
            if (menuX > 0)
                menuX -= 1;
            break;
        case 77: // Flecha derecha
            if (menuX < screenWidth - 1)
                menuX += 1;
            break;
        case 27: // Tecla Esc
            keyPressed = VK_F12;
            break;
    }
    setCursorPosition(menuX, menuY);
    cout << " ";
}

if (isKeyPressed(VK_F1)) {
    int baseLength, height;
    clearScreen();
    cout << "Ingrese la longitud de la base del triangulo: ";
    cin >> baseLength;
    cout << "Ingrese la altura del triangulo: ";
    cin >> height;
    drawTriangle(baseLength, height, x, y);
    keyPressed = _getch();
}
else if (isKeyPressed(VK_F2)) {
    int sideLength;
    clearScreen();
```

```
    cout << "Ingrese el lado del cuadrado: ";
    cin >> sideLength;
    drawSquare(sideLength, x, y);
    keyPressed = _getch();
}
else if (isKeyPressed(VK_F3)) {
    int width, height;
    clearScreen();
    cout << "Ingrese el ancho del rectangulo: ";
    cin >> width;
    cout << "Ingrese la altura del rectangulo: ";
    cin >> height;
    drawRectangle(width, height, x, y);
    keyPressed = _getch();
}
else if (isKeyPressed(VK_F4)) {
    int radius;
    clearScreen();
    cout << "Ingrese el radio del circulo: ";
    cin >> radius;
    drawCircle(radius, x, y);
    keyPressed = _getch();
}
else if (isKeyPressed(VK_F5)) {
    int length;
    string orientation;
    clearScreen();
    cout << "Ingrese la longitud de la línea: ";
    cin >> length;
```


```
            cout << "Ingrese la orientación (horizontal/vertical/diagonal): ";
            cin >> orientation;
            drawLine(length, orientation, x, y);
            keyPressed = _getch();
        }
        if (isKeyPressed(VK_F6)) {
            int sideLength;
            clearScreen();
            cout << "Ingrese la longitud de un lado del rombo: ";
            cin >> sideLength;
            drawDiamond(sideLength, x, y);
            keyPressed = _getch();
        }
        else if (isKeyPressed(VK_F7)) {
            int sideLength;
            clearScreen();
            cout << "Ingrese la longitud de un lado del hexagono: ";
            cin >> sideLength;
            drawHexagon(sideLength, x, y);
            keyPressed = _getch();
        }
    } while (keyPressed != VK_F12);
}

Sleep(20);
} while (keyPressed != 27);

return 0;
}
```

Conclusiones

El programa de graficación de figuras geométricas proporciona una herramienta simple pero efectiva para dibujar diversas figuras en una pantalla circular. Con una interfaz intuitiva y características de personalización, ofrece una experiencia de usuario flexible y satisfactoria. Con futuras mejoras y optimizaciones, este programa puede expandirse para satisfacer necesidades adicionales y brindar una funcionalidad aún más avanzada.