

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTÍN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	Tecnología de Objetos				
TÍTULO DE LA PRÁCTICA:	Programación Paralela (Threads)				
NÚMERO DE PRÁCTICA:	04	AÑO LECTIVO:	2025	NRO. SEMESTRE:	VI
FECHA DE PRESENTACIÓN	04/10/25	HORA DE PRESENTACIÓN	23:59		
INTEGRANTE (s): Boza Portilla Yordano Hernan Quispe Paucar Josue Claudio				NOTA:	
DOCENTE(s): Corrales Delgado Carlo Jose Luis					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <ol style="list-style-type: none"> 1. Para cualquier función definida y los puntos a y b. Hallar la integral utilizando el método del trapecio por aproximación. 2. Utilizar Programación Orientada a Objetos 3. Utilizar threads para los hilos. 4. Investigar el uso de Pool de Threads <p>Problema propuesto: Sea un función cualquier, por ejemplo: $f(x) = 2x^2 + 3x + \frac{1}{2}$ y los puntos $a = 2$ y $b = 20$ Hallar el área bajo la curva en el primer cuadrante, utilizando el método del trapecio. Utilice Java, C++ y Go para resolver el problema</p> <p>Método del Trapecio El método del trapecio es una técnica numérica para aproximar el valor de una integral definida cuando no se puede calcular de forma analítica.</p> $\int_a^b f(x) dx \sim \frac{h}{2} [f(a) + 2f(a+h) + 2f(a+2h) + \dots + f(b)]$

Donde:

$h = (b-a)/2$ es el ancho de cada subintervalo.

n = número de trapecios

Implementación en Java de manera secuencial

```
1 public class Trapecio{
2
3     public interface Funcion{
4         double eval(double x);
5     }
6
7     public static class FuncionEjemplo implements Funcion{
8         @Override
9         public double eval(double x){
10             return 2.0*x*x+3.0*x+0.5;
11         }
12     }
13 }
```

Se define la clase Trapecio, en esta se redacta toda la lógica y código que se usará para resolver el problema propuesto. Se crea la interfaz Función, ésta representa una función matemática genérica, cualquier clase que la use deberá definir cómo se evalúa la función. La clase estática FunciónEjemplo es una implementación concreta de Función, donde se define la función brindada en el problema.

```
14 public static class Integrador{
15     private final Funcion f;
16     private final double a, b;
17     private final long n;
18
19     public Integrador(Funcion f, double a, double b, long n){
20         this.f = f;
21         this.a = a;
22         this.b = b;
23         this.n = n;
24     }
25
26     public double integrar() {
27         double h = (b - a) / (double) n;
28         double sum = 0.5 * (f.eval(a) + f.eval(b));
29         for (long i = 1; i < n; i++) {
30             double x = a + i * h;
31             sum += f.eval(x);
32         }
33         return sum * h;
34     }
35 }
```

La clase Integrador es responsable de calcular la integral de una función f entre los límites a y b, dividiendo el intervalo en n subintervalos que son los trapecios. Su constructor recibe la función, el límite inferior a, el superior b y el número de subintervalos n. El método integrar() se calcula h y luego

inicia la suma con la mitad de los extremos $f(a)$ y $f(b)$. El bucle Multiplica la suma por el ancho de cada trapecio para obtener la aproximación de la integral.

```
37     public static void main(String[] args) {
38         long n = 1000000;
39         Funcion f = new FuncionEjemplo();
40         double a = 2.0, b = 20.0;
41
42         Integrador integrador = new Integrador(f, a, b, n);
43         long t0 = System.currentTimeMillis();
44         double result = integrador.integrar();
45         long t1 = System.currentTimeMillis();
46
47         System.out.printf(format:"Integral aproximada (Trapezio) = %.10f%n", result);
48         System.out.printf(format:"Subintervalos = %d, Tiempo = %d ms%n", n, (t1 - t0));
49     }
50 }
```

Finalmente, se definen los parámetros de interacción en el método principal que es el main.

- $a = 2.0$ (límite inferior)
- $b = 20.0$ (límite superior)
- $n = 1000000$ (subintervalos), se hacen diferentes pruebas

Crea una instancia de la función a integrar, Crea un objeto Integrador con la función y los parámetros, Mide el tiempo antes y después de la integración para ver el rendimiento, imprime el resultado aproximado y el tiempo total.

Ejecución

Con 1 trapecio

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 7875.0000000000
Subintervalos = 1, Tiempo = 0 ms
```

Con 10 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 5950.4400000000
Subintervalos = 10, Tiempo = 0 ms
```

Con 100 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 5931.1944000000
Subintervalos = 100, Tiempo = 0 ms
```

Con 1000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 5931.0019440000
Subintervalos = 1000, Tiempo = 0 ms
```

Con 10000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 5931.0000194400
Subintervalos = 10000, Tiempo = 0 ms
```

Con 100000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 5931.0000001943
Subintervalos = 100000, Tiempo = 3 ms
```

Con 1000000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE"
Integral aproximada (Trapezio) = 5931.00000000019
Subintervalos = 1000000, Tiempo = 6 ms
```

Implementación en Java con hilos

```
1  import java.util.*;
2  import java.util.concurrent.*;
3
4  public class TrapecioPool {
5
6      public static double f(double x) {
7          return 2.0 * x * x + 3.0 * x + 0.5;
8      }
9  }
```

Para empezar, usamos las librerías necesarias para usar List, ArrayList y herramientas para programación concurrente. Se crea la clase TrapecioPool y en la función f se coloca la función a evaluar.

```
10  public static class IntegrarParcial implements Callable<Double> {
11      private final double a, h;
12      private final long inicio, fin;
13      public IntegrarParcial(double a, double h, long inicio, long fin) {
14          this.a = a; this.h = h; this.inicio = inicio; this.fin = fin;
15      }
16
17      @Override
18      public Double call() {
19          double suma = 0.0;
20          for (long i = inicio; i <= fin; i++){
21              double x = a + i * h;
22              suma += f(x);
23          }
24          return suma;
25      }
```

Esta clase representa una tarea paralela que calcula una parte de la integral.

- a: límite inferior global de integración.
- h: ancho del subintervalo.
- inicio, fin: índices de los puntos que este hilo procesa.
- El constructor inicia los valores que delimitan el trabajo del hilo.
- En el método Call(), cada hilo calcula la suma parcial de $f(x_i)f(x_{i+1})f(x_i)$ entre los índices inicio y fin y devuelve la suma al hilo principal (usando Future<Double>). Esto permite dividir el trabajo en partes iguales.

```
28 public static double integrarPorTrapezioParalelo(double a, double b, long n, int numHilos)
29     throws InterruptedException, ExecutionException {
30     double h = (b - a) / n;
31     double sumaTotal = 0.5 * (f(a) + f(b));
32     ExecutorService pool = Executors.newFixedThreadPool(numHilos);
33     List<Future<Double>> resultados = new ArrayList<>();
34     long pasosPorHilo = n / numHilos;
35
36     for (int i = 0; i < numHilos; i++) {
37         long inicio = i * pasosPorHilo + 1;
38         long fin = (i == numHilos - 1) ? (n - 1) : ((i + 1) * pasosPorHilo);
39         resultados.add(pool.submit(new IntegrarParcial(a, h, inicio, fin)));
40     }
41
42     for (Future<Double> futuro : resultados) {
43         sumaTotal += futuro.get();
44     }
45
46     pool.shutdown();
47     return sumaTotal * h;
48 }
```

Esta función calcula la integral mediante el método del trapezio, distribuyendo el trabajo entre varios hilos. Se define a “h” como el ancho de cada subintervalo, luego “sumaTotal” se inicia con los extremos $f(a)$ y $f(b)$, como en la fórmula del trapezio. Luego de esto:

- ExecutorService: gestiona un grupo de hilos (thread pool).
- Future<Double> guarda el resultado que cada hilo devolverá cuando termine su tarea.
- Cada hilo integra un rango de subintervalos.

El último hilo toma cualquier resto para cubrir todo el intervalo, pool.submit(...) lanza la tarea y devuelve un Future con su resultado. Antes de terminar, futuro.get() espera que cada hilo termine y obtiene su resultado parcial, luego se van sumando todas las sumas parciales a sumaTotal.

Finalmente shutdown() cierra el pool de hilos una vez terminadas las tareas y se devuelve la integral final multiplicando por el ancho h.

```
Run | Debug | Run main | Debug main
50 public static void main(String[] args) throws InterruptedException, ExecutionException {
51     double a = 2.0, b = 20.0;
52     long n = 1000000;
53     int numHilos = 12;
54     System.out.println("Subintervalos: " + n);
55     System.out.println("Numero de hilos: " + numHilos);
56     long inicio = System.currentTimeMillis();
57     double resultado = integrarPorTrapezioParalelo(a, b, n, numHilos);
58     long fin = System.currentTimeMillis();
59     System.out.printf(format:"Resultado aproximado = %.15f%n", resultado);
60     System.out.printf(format:"Tiempo total = %d ms%n", (fin - inicio));
61 }
62 }
63
```

Ejecución

Con 1 trapecio

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to
Subintervalos: 1
Numero de hilos: 4
Resultado aproximado = 7875.000000000000000
Tiempo total = 6 ms
```

Con 10 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to
Subintervalos: 10
Numero de hilos: 4
Resultado aproximado = 5950.4400000000000500
Tiempo total = 5 ms
```

Con 100 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to
Subintervalos: 100
Numero de hilos: 4
Resultado aproximado = 5931.1944000000000000
Tiempo total = 5 ms
```

Con 1000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to
Subintervalos: 1000
Numero de hilos: 4
Resultado aproximado = 5931.001943999997000
Tiempo total = 6 ms
```

Con 10000 trapecios

Con 100000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to"
Subintervalos: 10000
Numero de hilos: 4
Resultado aproximado = 5931.000019439997000
Tiempo total = 6 ms
```

Con 1000000 trapecios

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to"
Subintervalos: 100000
Numero de hilos: 4
Resultado aproximado = 5931.000000194411000
Tiempo total = 8 ms
```

```
[Running] cd "c:\Users\ASUS\Documents\UNSA\6to"
Subintervalos: 1000000
Numero de hilos: 4
Resultado aproximado = 5931.000000001932000
Tiempo total = 18 ms
```

Implementación en C++ de manera secuencial

```
C++ trapecio.cpp > main()
1  #include <iostream>
2  #include <cmath>
3  #include <chrono>
4  #include <vector>
5  using namespace std;
6
7  class Funcion {
8  public:
9      virtual double evaluar(double x) const = 0;
10     virtual ~Funcion() = default;
11 };
```

En primer lugar se creo una clase abstracta llamada Funcion, esta clase contiene un método virtual puro evaluar(double x), que obliga a las clases derivadas a implementarlo. Por otro lado ~Funcion() = default, asegura que se destruya correctamente cualquier objeto derivado.

```
13 class FuncionCuadratica : public Funcion {
14 public:
15     double evaluar(double x) const override {
16         return 2.0 * x * x + 3.0 * x + 0.5;
17     }
18 };
```

Posteriormente se creo la clase FuncionCuadratica que hereda de la clase Funcion. En esta clase se representa la función específica que queremos integrar o hallar su área bajo la curva, en este caso sería $f(x) = 2x^2 + 3x + 0.5$

```
19
20 class IntegradorTrapezio {
21 private:
22     double a, b;
23     long n;
24     const Funcion& funcion;
25
26 public:
27     IntegradorTrapezio(double a_, double b_, long n_, const Funcion& funcion_)
28         : a(a_), b(b_), n(n_), funcion(funcion_) {}
29
30     double integrar() const {
31         double h = (b - a) / n;
32         double sumaTotal = 0.5 * (funcion.evaluar(a) + funcion.evaluar(b));
33
34         for (long i = 1; i < n; ++i) {
35             double x = a + i * h;
36             sumaTotal += funcion.evaluar(x);
37         }
38
39         return sumaTotal * h;
40     }
41 };
```

Asimismo, se creo la clase IntegradorTrapezio, en donde:

- Se definen sus atributos a y b son los limites de integración, n es el número de subintervalos(trapezios) y funcion, hace referencia a un objeto de tipo Funcion para poder evaluar la función que se quiere integrar
- Se define el constructor que inicializa los valores de los límites, número de trapezios y la función a integrar.
- Se crea el método integrar, que primero calcula la longitud de cada subintervalo, después inicializa la suma usando los extremos, según la fórmula del trapezio, luego itera sobre todos los puntos internos del intervalo, sumando el valor de la función en cada punto.Finalmente, multiplica la suma total por h para obtener la integral aproximada.


```
int main() {
    double a = 2.0;
    double b = 20.0;
    long n = 4;

    FuncionCuadratica funcion;
    IntegradorTrapezio integrador(a, b, n, funcion);

    auto inicio = chrono::high_resolution_clock::now();
    double resultado = integrador.integrar();
    auto fin = chrono::high_resolution_clock::now();

    chrono::duration<double, milli> duracion = fin - inicio;

    cout.precision(14);
    cout << fixed;
    cout << "Integral aproximada (Trapezio) = " << resultado << endl;
    cout << "Subintervalos = " << n << ", Tiempo = " << duracion.count() << " ms" << endl;

    return 0;
}
```

Finalmente, en la función main, primero se definen los límites de integración a y b y el número de subintervalos n que se usarán para el método del trapezio. Luego se crea un objeto de la clase FuncionCuadratica, que representa la función que se va a integrar, y se pasa a un objeto de IntegradorTrapezio junto con los límites y el número de trapecios. A continuación, se mide el tiempo de ejecución iniciando un cronómetro antes de llamar al método integrar() y deteniéndolo después, para calcular la duración en milisegundos. Finalmente, se imprime en pantalla el valor aproximado de la integral, el número de subintervalos utilizados y el tiempo que tomó el cálculo, mostrando el resultado con precisión de 14 decimales.

Ejecución

Con 1 trapezio

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 7875.00000000000000
Subintervalos = 1, Tiempo = 0.00020000000000 ms
```

Con 10 trapecios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapezio.cpp -o trapezio
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 5950.43999999999960
Subintervalos = 10, Tiempo = 0.00060000000000 ms
```

Con 100 trapecios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapezio.cpp -o trapezio
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 5931.194399999999850
Subintervalos = 100, Tiempo = 0.00180000000000 ms
```

Con 1000 trapecios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> g++ trapezio.cpp -o trapezio
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 5931.00194399999327
Subintervalos = 1000, Tiempo = 0.01400000000000 ms
```

Con 10000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> g++ trapezio.cpp -o trapezio
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 5931.00001944001360
Subintervalos = 10000, Tiempo = 0.08670000000000 ms
```

Con 100000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> g++ trapezio.cpp -o trapezio
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 5931.00000019433173
Subintervalos = 100000, Tiempo = 1.60810000000000 ms
```

Con 1000000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> g++ trapezio.cpp -o trapezio
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_T0\Pruebas> ./trapezio
Integral aproximada (Trapezio) = 5931.00000000192722
Subintervalos = 1000000, Tiempo = 14.54320000000000 ms
```

Implementación en C++ con hilos

```
C++ trapezioPool.cpp > IntegradorTrapezioParalelo > integrar() const
1  #include <iostream>
2  #include <cmath>
3  #include <thread>
4  #include <vector>
5  #include <chrono>
6  using namespace std;
7
8  class Funcion {
9  public:
10     virtual double evaluar(double x) const {
11         return 2.0 * x * x + 3.0 * x + 0.5;
12     }
13
14     virtual ~Funcion() = default;
15 };
```

En primer lugar, esta parte se define una clase Funcion en C++ que representa una función matemática genérica. La clase contiene un método evaluar(double x) que devuelve el valor de la función cuadrática $f(x) = 2x^2 + 3x + 0.5$ para un valor dado de x, y está marcado como virtual para que pueda ser sobrescrito por clases derivadas si se desea definir otra función diferente. Además, incluye un destructor virtual `~Funcion() = default;` para asegurar que los objetos derivados se destruyan correctamente cuando se manejan a través de punteros o referencias a la clase base. Esto permite que la clase sirva como base para integrar o evaluar distintas funciones de manera polimórfica.

```
17 class IntegradorTrapezioParalelo {
18 private:
19     double a, b;
20     long n;
21     int numHilos;
22     const Funcion& funcion;
23
24     void integrarParcial(double h, long inicio, long fin, long double& resultadoParcial) const {
25         long double suma = 0.0;
26         for (long i = inicio; i <= fin; ++i) {
27             double x = a + i * h;
28             suma += funcion.evaluar(x);
29         }
30         resultadoParcial = suma;
31     }
32 }
```

Seguidamente, se creo una clase IntegradorTrapezioParalelo en C++ que se utiliza para aproximar integrales mediante el método del trapezio aprovechando múltiples hilos para paralelizar el cálculo. La clase almacena los límites de integración a y b, el número de subintervalos n, el número de hilos numHilos y una referencia a un objeto Funcion que representa la función a integrar. Además, define un método privado integrarParcial que calcula la suma de los valores de la función en un rango específico de subintervalos, desde inicio hasta fin, multiplicando cada índice por el ancho h del subintervalo y acumulando el resultado en resultadoParcial. Este método permite dividir el trabajo entre varios hilos para acelerar el cálculo de la integral.

```
33 public:
34     IntegradorTrapezioParalelo(double a_, double b_, long n_, int numHilos_, const Funcion& funcion_)
35         : a(a_), b(b_), n(n_), numHilos(numHilos_), funcion(funcion_) {}
36
37     long double integrar() const {
38         double h = (b - a) / n;
39         long double sumaTotal = 0.5L * (funcion.evaluar(a) + funcion.evaluar(b));
40
41         vector<thread> hilos(numHilos);
42         vector<long double> resultados(numHilos, 0.0);
43
44         long pasosPorHilo = n / numHilos;
45
46         for (int i = 0; i < numHilos; ++i) {
47             long inicio = i * pasosPorHilo + 1;
48             long fin = (i == numHilos - 1) ? (n - 1) : ((i + 1) * pasosPorHilo);
49
50             hilos[i] = thread(&IntegradorTrapezioParalelo::integrarParcial, this,
51                             h, inicio, fin, ref(resultados[i]));
52         }
53
54         for (int i = 0; i < numHilos; ++i) {
55             hilos[i].join();
56             sumaTotal += resultados[i];
57         }
58
59         return sumaTotal * h;
60     }
```

Luego, se definió la parte pública de la clase IntegradorTrapezioParalelo. Primero, el constructor inicializa los límites de integración a y b, el número de subintervalos n, la cantidad de hilos numHilos y la función a integrar funcion. Luego, el método integrar calcula la integral aproximada utilizando paralelización: se determina el ancho de cada subintervalo h y se inicia la suma con la contribución de los extremos de la integral. Se crean vectores para los hilos y para almacenar los resultados parciales de cada hilo, y se divide el total de pasos entre los hilos. Cada hilo ejecuta el método integrarParcial sobre su rango asignado de subintervalos, almacenando la suma parcial en el vector correspondiente. Finalmente, se espera a que todos los hilos terminen con join, se suman los resultados parciales a la suma total y se multiplica por h para obtener la integral aproximada.

```
63  int main() {
64      double a = 2.0, b = 20.0;
65      long n = 1'000'000;
66      int numHilos = 4;
67
68      Funcion funcion;
69
70      IntegradorTrapezioParalelo integrador(a, b, n, numHilos, funcion);
71
72      auto inicio = chrono::high_resolution_clock::now();
73      long double resultado = integrador.integrar();
74      auto fin = chrono::high_resolution_clock::now();
75
76      chrono::duration<double, milli> duracion = fin - inicio;
77
78      cout.precision(15);
79      cout << "Integral aproximada (Trapezio paralelo) = " << (double)resultado << endl;
80      cout << "Subintervalos = " << n << ", Hilos = " << numHilos
81          << ", Tiempo = " << duracion.count() << " ms" << endl;
82
83      return 0;
84  }
```

Finalmente, en la función main se definen los límites de integración a y b, el número de subintervalos n y la cantidad de hilos numHilos que se utilizarán para calcular la integral de forma paralela. Se crea un objeto Funcion que representa la función a integrar, y se pasa junto con los parámetros anteriores a un objeto IntegradorTrapezioParalelo. A continuación, se inicia un cronómetro antes de llamar al método integrar() para medir el tiempo de ejecución, y se detiene justo después de obtener el resultado. Finalmente, se imprime en pantalla la integral aproximada (convertida a double), el número de subintervalos, la cantidad de hilos usados y el tiempo total de cálculo en milisegundos, mostrando el resultado con una precisión de 15 decimales.

Ejecución

Con 1 trapezio

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 7875
Subintervalos = 1, Hilos = 4, Tiempo = 0.7354 ms
```

Con 10 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 5950.44
Subintervalos = 10, Hilos = 4, Tiempo = 1.376 ms
```

Con 100 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 5931.1944
Subintervalos = 100, Hilos = 4, Tiempo = 0.8916 ms
```

Con 1000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 5931.001944
Subintervalos = 1000, Hilos = 4, Tiempo = 0.6308 ms
```

Con 10000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 5931.00001944
Subintervalos = 10000, Hilos = 4, Tiempo = 1.4308 ms
```

Con 100000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 5931.0000001944
Subintervalos = 100000, Hilos = 4, Tiempo = 1.6905 ms
```

Con 1000000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> g++ trapecioPool.cpp -o trapecioPool
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> ./trapecioPool
Integral aproximada (Trapezio paralelo) = 5931.00000000194
Subintervalos = 1000000, Hilos = 4, Tiempo = 7.4308 ms
```

Implementación en Go de manera secuencial

```
1  package main
2
3  import (
4      "fmt"
5      "time"
6  )
7
8  type Funcion interface {
9      Evaluar(x float64) float64
10 }
11
12 type FuncionCuadratica struct{}
13
14 func (f FuncionCuadratica) Evaluar(x float64) float64 {
15     return 2.0*x*x + 3.0*x + 0.5
16 }
```

En primer lugar, este fragmento en Go define una interfaz `Funcion` que obliga a cualquier tipo que la implemente a tener un método `Evaluar(x float64) float64`, el cual devuelve el valor de la función para un valor dado x . Luego se declara una estructura `FuncionCuadratica` que implementa esta interfaz mediante el método `Evaluar`, calculando y retornando el valor de la función cuadrática $f(x) = 2x^2 + 3x + 0.5$. Esto permite que `FuncionCuadratica` pueda ser utilizada de manera genérica en cualquier algoritmo que trabaje con la interfaz `Funcion`, como un integrador numérico.

```
18 type IntegradorTrapezio struct {
19     a, b float64
20     n    int64
21     funcion Funcion
22 }
```

Seguidamente, se define la estructura `IntegradorTrapezio` en Go, que representa un integrador numérico usando el método del trapecio. Esta estructura contiene los límites de integración a y b , el número de subintervalos n que se usarán para aproximar la integral, y un campo `funcion` de tipo `Funcion`, que permite evaluar cualquier función que implemente la interfaz `Funcion`. Esto proporciona una forma genérica de calcular integrales para distintas funciones matemáticas.


```
24 func (it IntegradorTrapezio) Integrar() float64 {
25     h := (it.b - it.a) / float64(it.n)
26     sumaTotal := 0.5 * (it.funcion.Evaluar(it.a) + it.funcion.Evaluar(it.b))
27
28     for i := int64(1); i < it.n; i++ {
29         x := it.a + float64(i)*h
30         sumaTotal += it.funcion.Evaluar(x)
31     }
32
33     return sumaTotal * h
34 }
```

Asimismo, se define el método Integrar para la estructura IntegradorTrapezio, que calcula la integral aproximada de la función usando el método del trapecio. Primero, se determina el ancho de cada subintervalo h dividiendo la longitud del intervalo entre el número de subintervalos n . Luego, se inicializa la suma total con la contribución de los extremos de la integral, multiplicando por 0.5 según la fórmula del trapecio. A continuación, se recorre cada subintervalo interno, evaluando la función en cada punto y sumando los resultados a sumaTotal. Finalmente, se multiplica la suma acumulada por h para obtener la aproximación de la integral completa.

```
36 func main() {
37     a := 2.0
38     b := 20.0
39     n := int64(1_000_000)
40
41     funcion := FuncionCuadratica{}
42     integrador := IntegradorTrapezio{
43         a: a,
44         b: b,
45         n: n,
46         funcion: funcion,
47     }
48
49     inicio := time.Now()
50     resultado := integrador.Integrar()
51     duracion := time.Since(inicio)
52
53     fmt.Printf("Integral aproximada (Trapezio) = %.14f\n", resultado)
54     fmt.Printf("Subintervalos = %d, Tiempo = %.3f ms\n", n, float64(duracion.Microseconds())/1000.0)
55 }
```

Finalmente, en la función main se definen los límites de integración a y b y el número de subintervalos n para calcular la integral. Se crea un objeto FuncionCuadratica y se utiliza para inicializar un IntegradorTrapezio con los parámetros correspondientes. Luego, se registra el tiempo de inicio antes de llamar al método Integrar y se calcula la duración transcurrida después de obtener el resultado. Por último, se imprime en pantalla la integral aproximada con 14 decimales, junto con el número de subintervalos y el tiempo de ejecución en milisegundos.

Ejecución

Con 1 trapecio

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapecio.go
Integral aproximada (Trapezio) = 7875.000000000000
Subintervalos = 1, Tiempo = 0.000 ms
```

Con 10 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapezio.go
Integral aproximada (Trapezio) = 5950.4399999999996
Subintervalos = 10, Tiempo = 0.000 ms
```

Con 100 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapezio.go
Integral aproximada (Trapezio) = 5931.1943999999985
Subintervalos = 100, Tiempo = 0.000 ms
```

Con 1000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapezio.go
Integral aproximada (Trapezio) = 5931.00194399999327
Subintervalos = 1000, Tiempo = 0.000 ms
```

Con 10000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapezio.go
Integral aproximada (Trapezio) = 5931.00001944001360
Subintervalos = 10000, Tiempo = 0.000 ms
```

Con 100000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapezio.go
Integral aproximada (Trapezio) = 5931.00000019433173
Subintervalos = 100000, Tiempo = 0.584 ms
```

Con 1000000 trapezios

```
PS E:\Ing. Sistemas\3er_año\Semestre_VI\Lab_TO\Pruebas> go run trapezio.go
Integral aproximada (Trapezio) = 5931.00000000192722
Subintervalos = 1000000, Tiempo = 3.922 ms
```

Implementación en Go con hilos

```
8
9  type Funcion interface {
10     |   Evaluar(x float64) float64
11     | }
12
13  type FuncionCuadratica struct{}
14
15  func (f FuncionCuadratica) Evaluar(x float64) float64 {
16     |   return 2.0*x*x + 3.0*x + 0.5
17     | }
```

Funcion es una interfaz que define un método Evaluar(x) para cualquier función matemática. FuncionCuadratica implementa esta interfaz con la función.


```

19 type IntegradorTrapezioParalelo struct {
20     a, b    float64
21     n      int64
22     numHilos int
23     funcion  Funcion
24 }
25
26 func (it IntegradorTrapezioParalelo) integrarParcial(h float64, inicio, fin int64, wg *sync.WaitGroup, resultado *float64, mutex *sync.Mutex) {
27     defer wg.Done()
28
29     suma := 0.0
30     for i := inicio; i <= fin; i++ {
31         x := it.a + float64(i)*h
32         suma += it.funcion.Evaluar(x)
33     }
34
35     mutex.Lock()
36     *resultado += suma
37     mutex.Unlock()
38 }

```

La estructura del integrador: Guarda los parámetros del problema:

- a y b: límites de integración.
- n: número total de subdivisiones (trapezios).
- numHilos: cantidad de hilos (goroutines).
- funcion: la función a integrar.

Luego de esto, Cada hilo (goroutine) calcula una suma parcial de la función en un rango [inicio, fin], wg (WaitGroup) coordina los hilos para que el programa espere a que todos terminen, mutex asegura que el acceso a resultado sea seguro (evita condiciones de carrera). Cada hilo acumula su suma y la agrega de forma sincronizada al resultado total.

```

40 func (it IntegradorTrapezioParalelo) Integrar() float64 {
41     h := (it.b - it.a) / float64(it.n)
42     sumaTotal := 0.5 * (it.funcion.Evaluar(it.a) + it.funcion.Evaluar(it.b))
43
44     var resultado float64 = 0.0
45     var wg sync.WaitGroup
46     var mutex sync.Mutex
47
48     pasosPorHilo := it.n / int64(it.numHilos)
49
50     for i := 0; i < it.numHilos; i++ {
51         inicio := int64(i)*pasosPorHilo + 1
52         var fin int64
53         if i == it.numHilos-1 {
54             fin = it.n - 1
55         } else {
56             fin = (int64(i+1) * pasosPorHilo)
57         }
58
59         wg.Add(1)
60         go it.integrarParcial(h, inicio, fin, &wg, &resultado, &mutex)
61     }
62
63     wg.Wait()
64     sumaTotal += resultado
65     return sumaTotal * h
66 }
67

```

El método principal de integración Calcula el ancho del trapecio h . Empieza con los extremos de la integral ($f(a)$ y $f(b)$) ponderados con $\frac{1}{2}$. Cada hilo calcula una parte de la integral. Espera a que todos los hilos terminen y multiplica por h para obtener la integral total.

```
68 func main() {
69     a := 2.0
70     b := 20.0
71     n := int64(1_000_000)
72     numHilos := 12
73
74     funcion := FuncionCuadratica{}
75     integrador := IntegradorTrapezioParalelo{
76         a:      a,
77         b:      b,
78         n:      n,
79         numHilos: numHilos,
80         funcion: funcion,
81     }
82
83     inicio := time.Now()
84     resultado := integrador.Integrar()
85     duracion := time.Since(inicio)
86
87     fmt.Printf("Integral aproximada (Trapezio paralelo) = %.15f\n", resultado)
88     fmt.Printf("Subintervalos = %d, Hilos = %d, Tiempo = %.3f ms\n", n, numHilos, float64(duracion.Microseconds())/1000.0)
89 }
90
```

Ejecución

Con 1 trapecio

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tecno
Integral aproximada (Trapezio paralelo) = 7875.000000000000000
Subintervalos = 1, Hilos = 4, Tiempo = 0.000 ms
```

Con 10 trapecios

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tecno
Integral aproximada (Trapezio paralelo) = 5950.440000000000509
Subintervalos = 10, Hilos = 4, Tiempo = 0.000 ms
```



Con 100 trapecios

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tecno
Integral aproximada (Trapezio paralelo) = 5931.194400000000314
Subintervalos = 100, Hilos = 4, Tiempo = 0.000 ms
```

Con 1000 trapecios

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tecno
Integral aproximada (Trapezio paralelo) = 5931.001943999996911
Subintervalos = 1000, Hilos = 4, Tiempo = 0.000 ms
```

Con 10000 trapecios

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTÍN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 19</p>

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tecno
Integral aproximada (Trapezio paralelo) = 5931.000019439997232
Subintervalos = 10000, Hilos = 4, Tiempo = 0.000 ms
```

Con 100000 trapecios

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tec
Integral aproximada (Trapezio paralelo) = 5931.000000194410859
Subintervalos = 100000, Hilos = 4, Tiempo = 0.748 ms
```

Con 1000000 trapecios

```
[Running] go run "c:\Users\ASUS\Documents\UNSA\6to SEMESTRE\Tecno
Integral aproximada (Trapezio paralelo) = 5931.000000001931767
Subintervalos = 1000000, Hilos = 4, Tiempo = 0.532 ms
```

II. CUESTIONARIO

1. ¿Cuál de los LP posee ventajas para programar paralelamente?

Cuando se analiza la capacidad de los lenguajes de programación utilizados —Go, Java y C++— para programar en paralelo, se observa que cada uno tiene sus ventajas y limitaciones.

C++ ofrece un control muy fino sobre los hilos y la memoria a través de la biblioteca estándar (`<thread>`), lo que permite aprovechar al máximo el hardware del sistema. Sin embargo, esta flexibilidad también implica una mayor complejidad, ya que el programador debe manejar explícitamente la sincronización, la seguridad de los datos compartidos y posibles condiciones de carrera. Esto puede hacer que la programación paralela en C++ sea más potente, pero también más propensa a errores si no se aplica cuidadosamente.

Java cuenta con un modelo de hilos relativamente sencillo, soportado directamente por la máquina virtual (JVM), y proporciona utilidades de alto nivel como `ExecutorService` y `Future` para manejar tareas concurrentes. Esto facilita la escritura de programas paralelos seguros y más legibles que en C++, aunque puede generar una sobrecarga en tiempo de ejecución debido a la abstracción que introduce la JVM.

Go, por su parte, se destaca por su modelo de concurrencia integrado basado en goroutines y canales, que simplifica enormemente la programación paralela. Las goroutines son ligeras y se gestionan automáticamente por el runtime de Go, lo que permite crear miles de tareas concurrentes sin preocuparse por la complejidad de la gestión de hilos o sincronización manual. Este enfoque hace que Go posea claras ventajas para desarrollar aplicaciones concurrentes de manera sencilla, eficiente y segura, especialmente en comparación con la mayor complejidad de C++ y la sobrecarga relativa de Java.

En resumen, aunque C++ puede ofrecer el máximo control y Java simplifica ciertas tareas concurrentes, Go proporciona la ventaja más significativa para programar paralelamente, ya que combina eficiencia, facilidad de uso y un modelo de concurrencia moderno que reduce errores comunes en programación paralela.

2. ¿Para cada lenguaje elabore una tabla cuando usa Pool de Threads?

C++

Característica	Descripción
Implementación de Thread Pool	C++ estándar no tiene Thread Pool nativo; se requiere implementarlo manualmente o usar librerías externas (Boost, ThreadPool, etc.)
Creación de hilos	std::thread t(función)
Gestión del pool	Hay que controlar explícitamente la cola de tareas, los hilos y la sincronización
Ventajas	Control completo sobre los hilos y la memoria; máximo rendimiento
Desventajas	Más complejidad; riesgo de errores (condiciones de carrera, deadlocks)

Java

Característica	Descripción
Implementación de Thread Pool	Nativo con ExecutorService, ThreadPoolExecutor
Creación de hilos	ExecutorService executor = Executors.newFixedThreadPool(n)
Gestión del pool	Automática; se pueden enviar tareas con submit() y se manejan los hilos internamente
Ventajas	Fácil de usar; gestión automática de hilos; seguro para concurrencia
Desventajas	Menor control fino que C++; overhead de la JVM

Go

Característica	Descripción
Implementación de Thread Pool	No usa hilos directamente; se crean goroutines y se puede implementar un worker pool con canales
Creación de hilos	go función() para lanzar una goroutine ligera
Gestión del pool	Se puede usar un canal como cola de tareas y limitar el número de workers
Ventajas	Muy sencillo y eficiente; miles de goroutines ligeras; runtime gestiona hilos
Desventajas	Menor control sobre la planificación exacta de hilos a nivel de CPU

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA

1. Stroustrup, B. (2013). *The C++ Programming Language (4th ed.)*. Addison-Wesley.
2. <https://www.programarya.com/Cursos/C++/Estructuras-de-Datos/Punteros>
3. https://www.w3schools.com/cpp/cpp_pointers.asp