


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

## INFORME DE LABORATORIO

### (formato estudiante)

INFORMACIÓN BÁSICA					
<b>ASIGNATURA:</b>	Tecnología de Objetos				
<b>TÍTULO DE LA PRÁCTICA:</b>	Programar paralelamente usando threads.				
<b>NÚMERO DE PRÁCTICA:</b>	04	<b>AÑO LECTIVO:</b>	2025B	<b>NRO. SEMESTRE:</b>	
<b>FECHA DE PRESENTACIÓN</b>	04/10/2025	<b>HORA DE PRESENTACIÓN</b>	23:59		
<b>INTEGRANTE (s):</b> Cahui Benegas Anthony Ronaldo				<b>NOTA:</b>	
<b>DOCENTE(s):</b> Carlo Corrales					

SOLUCIÓN Y RESULTADOS
<p><b>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</b></p> <p>Sea un función cualquier, por ejemplo: <math>f(x) = 2x^2 + 3x + \frac{1}{2}</math>            y los puntos <math>a=2</math> y <math>b=20</math>            Hallar el área bajo la curva en el primer cuadrante, utilizando el método del trapecio.            En Clase:            Utilice Java para resolver el problema.            Utilice C++ para gestionar la memoria dinámicamente.            La próxima clase:            Utilice Go para resolver el problema.            LINK DE GITHUB: <a href="https://github.com/acahuib/teo/tree/main/lab04">https://github.com/acahuib/teo/tree/main/lab04</a></p> <p>Para todas las clases que se haran para java, c++ y go, se trabajaran en ase casi siempre a la funcion del trapecio para resolverlo que es:</p>

```
9
10 class Integrador extends Thread {
11     private int n;
12     private double a, b;
13     private double resultado;
14     private Funcion f;
15
16     public Integrador(int n, double a, double b, Funcion f) {
17         this.n = n;
18         this.a = a;
19         this.b = b;
20         this.f = f;
21     }
22
23     public void run() {
24         double h = (b - a) / n;
25         double suma = 0.0;
26         for (int i = 1; i < n; i++) {
27             double x = a + i * h;
28             suma += f.evaluar(x);
29         }
30         resultado = (h / 2) * (f.evaluar(a) + 2 * suma + f.evaluar(b));
31     }
32
33     public double getResultado() {
34         return resultado;
35     }
36 }
37
```

Para c++ se tiene:

```
7
8  class Funcion {
9  public:
10     double evaluar(double x) {
11         return 2 * x * x + 3 * x + 0.5;
12     }
13 };
14
15 class Integrador {
16     int n;
17     double a, b, resultado;
18     Funcion f;
19 public:
20     Integrador(int n, double a, double b): n(n), a(a), b(b), resultado(0) {}
21     void calcular() {
22         double h = (b - a) / n;
23         double suma = 0.0;
24         for (int i = 1; i < n; i++) {
25             double x = a + i * h;
26             suma += f.evaluar(x);
27         }
28         resultado = (h / 2) * (f.evaluar(a) + 2 * suma + f.evaluar(b));
29     }
30     double getResultado() { return resultado; }
31 };
```

Para go se tiene:

```
2
3  import (
4      "fmt"
5      "sync"
6  )
7
8  func f(x float64) float64 {
9      return 2*x*x + 3*x + 0.5
10 }
11
12 func integrar(n int, a, b float64, wg *sync.WaitGroup, resultado *float64) {
13     defer wg.Done()
14     h := (b - a) / float64(n)
15     suma := 0.0
16     for i := 1; i < n; i++ {
17         x := a + float64(i)*h
18         suma += f(x)
19     }
20     *resultado = (h / 2) * (f(a) + 2*suma + f(b))
21 }
```

Para ahora la clase de trapecio.java:

- Calcula el área bajo la curva  $f(x) = 2x^2 + 3x + 0.5$  desde  $a = 2$  hasta  $b = 20$ .
- Usa una clase Integrador que hereda de Thread, aplicando programación orientada a objetos (POO).
- Incrementa el número de trapecios  $n$  hasta que el resultado se repite (es decir, el valor de la integral converge).

```
37
38 public class trapecio {
    Run | Debug
39     public static void main(String[] args) throws InterruptedException {
40         Funcion f = new Funcion();
41         double anterior = -1;
42         int n = 1;
43
44         while (true) {
45             Integrador hilo = new Integrador(n, a:2, b:20, f);
46             hilo.start();
47             hilo.join();
48             double actual = hilo.getResultado();
49             System.out.println("n=" + n + " -> " + actual);
50             if (actual == anterior)
51                 break;
52             anterior = actual;
53             n++;
54         }
55     }
56 }
57
```

Para la clase trapecio.cpp:

- Implementa el método del trapecio de manera orientada a objetos (clases Funcion e Integrador).
- Usa new y delete para gestionar la memoria manualmente (requisito de C++).
- Crea un hilo con std::thread para realizar el cálculo de la integral.

```
32
33  int main() {
34      Funcion f;
35      double anterior = -1;
36      int n = 1;
37      while (true) {
38          Integrador* integ = new Integrador(n, 2, 20);
39          thread t(&Integrador::calcular, integ);
40          t.join();
41          double actual = integ->getResultado();
42          cout << "n=" << n << " -> " << actual << endl;
43          if (actual == anterior) break;
44          anterior = actual;
45          n++;
46          delete integ;
47      }
48      return 0;
49  }
50
```

Para trapezioGo se tiene:

- Calcula la integral con el mismo método, pero usando go integrar() para lanzar una goroutine.
- Usa sync.WaitGroup para esperar que termine la ejecución concurrente.
- Incrementa n hasta que la integral deja de cambiar.

```
23 func main() {
24     anterior := -1.0
25     n := 1
26     for {
27         var wg sync.WaitGroup
28         var res float64
29         wg.Add(1)
30         go integrar(n, 2, 20, &wg, &res)
31         wg.Wait()
32         fmt.Printf("n=%d -> %.5f\n", n, res)
33         if res == anterior {
34             break
35         }
36         anterior = res
37         n++
38     }
39 }
40
```

Para ahora el trapecioPool.java, se tiene:

- Reutiliza hilos en lugar de crear uno nuevo para cada cálculo.
- Usa un FixedThreadPool con 4 hilos y tareas Callable.
- Esto hace la simulación más eficiente en sistemas de 64 bits, donde los hilos son costosos de crear.

```
7 class TareaTrapecio implements Callable<Double> {
8     private int n; private double a, b; private Funcion f;
9     public TareaTrapecio(int n, double a, double b, Funcion f) {
10         this.n = n; this.a = a; this.b = b; this.f = f;
11     }
12     public Double call() {
13         double h = (b - a) / n;
14         double suma = 0.0;
15         for (int i = 1; i < n; i++) suma += f.evaluar(a + i * h);
16         return (h / 2) * (f.evaluar(a) + 2 * suma + f.evaluar(b));
17     }
18 }
19
20 public class TrapecioPool {
21     Run | Debug
22     public static void main(String[] args) throws Exception {
23         ExecutorService pool = Executors.newFixedThreadPool(nThreads:4);
24         Funcion f = new Funcion();
25         double anterior = -1; int n = 1;
26
27         while (true) {
28             Future<Double> resultado = pool.submit(new TareaTrapecio(n, a:2, b:20, f));
29             double actual = resultado.get();
30             System.out.println("n=" + n + " -> " + actual);
31             if (actual == anterior) break;
32             anterior = actual;
33             n++;
34         }
35         pool.shutdown();
36     }
37 }
```

Para trapecioPool.cpp se tiene:

- Usa `std::async` para lanzar tareas asíncronas (en lugar de crear manualmente threads).
- El sistema reutiliza internamente los hilos disponibles.
- Calcula el resultado y sincroniza con `future.get()`.



```
6  double f(double x) { return 2 * x * x + 3 * x + 0.5; }
7
8  double integrar(int n, double a, double b) {
9      double h = (b - a) / n;
10     double suma = 0.0;
11     for (int i = 1; i < n; i++) {
12         double x = a + i * h;
13         suma += f(x);
14     }
15     return (h / 2) * (f(a) + 2 * suma + f(b));
16 }
17
18 int main() {
19     double anterior = -1;
20     int n = 1;
21     while (true) {
22         future<double> fut = async(launch::async, integrar, n, 2.0, 20.0);
23         double actual = fut.get();
24         cout << "n=" << n << " -> " << actual << endl;
25         if (actual == anterior) break;
26         anterior = actual;
27         n++;
28     }
29     return 0;
30 }
31
```

Para trapecioPool.go se tiene:

- Crea un canal pool con capacidad 4, simulando un grupo de 4 trabajadores.
- Lanza goroutines que calculan la integral mientras el pool controla cuántas se ejecutan a la vez.
- Se detiene cuando el valor calculado se repite.

```

8 func f(x float64) float64 {
9     return 2*x*x + 3*x + 0.5
10 }
11
12 func integrar(n int, a, b float64, ch chan float64) {
13     h := (b - a) / float64(n)
14     suma := 0.0
15     for i := 1; i < n; i++ {
16         suma += f(a + float64(i)*h)
17     }
18     ch <- (h / 2) * (f(a) + 2*suma + f(b))
19 }
20
21 func main() {
22     var anterior float64 = -1
23     n := 1
24     pool := make(chan struct{}, 4) // Pool de 4 workers
25     for {
26         pool <- struct{}{}
27         ch := make(chan float64)
28         go func(n int) {
29             integrar(n, 2, 20, ch)
30             <-pool
31         }(n)
32         resultado := <-ch
33         fmt.Printf("n=%d -> %.5f\n", n, resultado)
34         if resultado == anterior {
35             break
36         }
37         anterior = resultado
38         n++
39     }

```

## II. SOLUCIÓN DEL CUESTIONARIO

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

El lenguaje que posee más ventajas para programar paralelamente es Go, porque tiene soporte nativo para concurrencia mediante goroutines y canales, que son más ligeros y eficientes que los threads tradicionales de Java o C++.

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

Lenguaje	Mecanismo de Pool de Threads	Ventajas principales
Java	ExecutorService y ThreadPoolExecutor	Fácil de implementar, gestiona automáticamente los hilos, buena estabilidad.
C++	std::thread combinado con std::async o librerías como thread pool	Permite control manual y eficiente de los recursos, aunque más complejo.



UNIVERSIDAD NACIONAL DE SAN AGUSTIN  
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS  
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA



**Formato:** Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

**Aprobación:** 2022/03/01

**Código:** GUIA-PRLE-001

**Página:** 11

Go

Uso de goroutines con worker  
pools y canales

Alta eficiencia, bajo consumo de  
memoria, gestión automática de  
conurrencia.