

Laboratorio 5

1. (18 pts.) Explica con tus propias palabras los siguientes términos:

a) **private**

Private especifica que cada hilo tiene su propia copia de la variable. Al utilizarse, se inicializan en cada hilo con un valor indefinido y no comparten los valores con otros hilos al final de la sección paralela.

b) **shared**

Shared indica que la variable se comparte entre todos los hilos. Esto permite que todos los hilos puedan leer y escribir sobre la misma variable.

c) **firstprivate**

Firstprivate es básicamente lo mismo que private, con la única diferencia de que las variables se inician con el valor que tenían antes de la región paralela.

d) **barrier**

Barrier actúa como barrera de sincronización, haciendo que todos los hilos tengan que alcanzar un punto específico antes de seguir con la ejecución.

e) **critical**

Define una sección de código que solo puede ejecutar un hilo a la vez. Esta se distingue de Atomic ya que se pueden utilizar varias líneas de código para tareas más complejas.

f) **atomic**

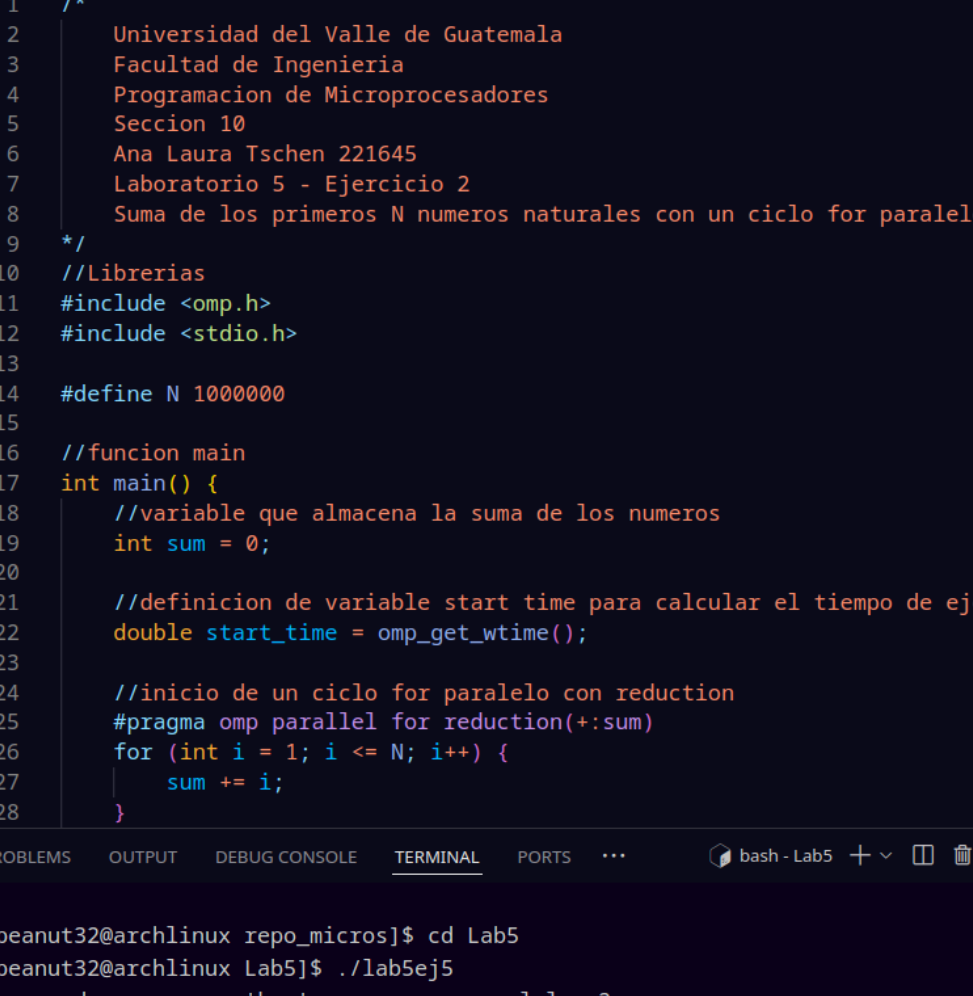
Atomic se utiliza para realizar acciones elementales, esta permite escribir, leer, o actualizar y se recomienda utilizar una línea de código.

Link repositorio de Github: https://github.com/tsc221645/Microprocesadores_CC2086_2024

2. (12 pts.) Escribe un programa en C que calcule la suma de los primeros N números naturales utilizando un ciclo for paralelo. Utiliza la cláusula reduction con + para acumular la suma en una variable compartida.

a) Define N como una constante grande, por ejemplo, $N = 1000000$.

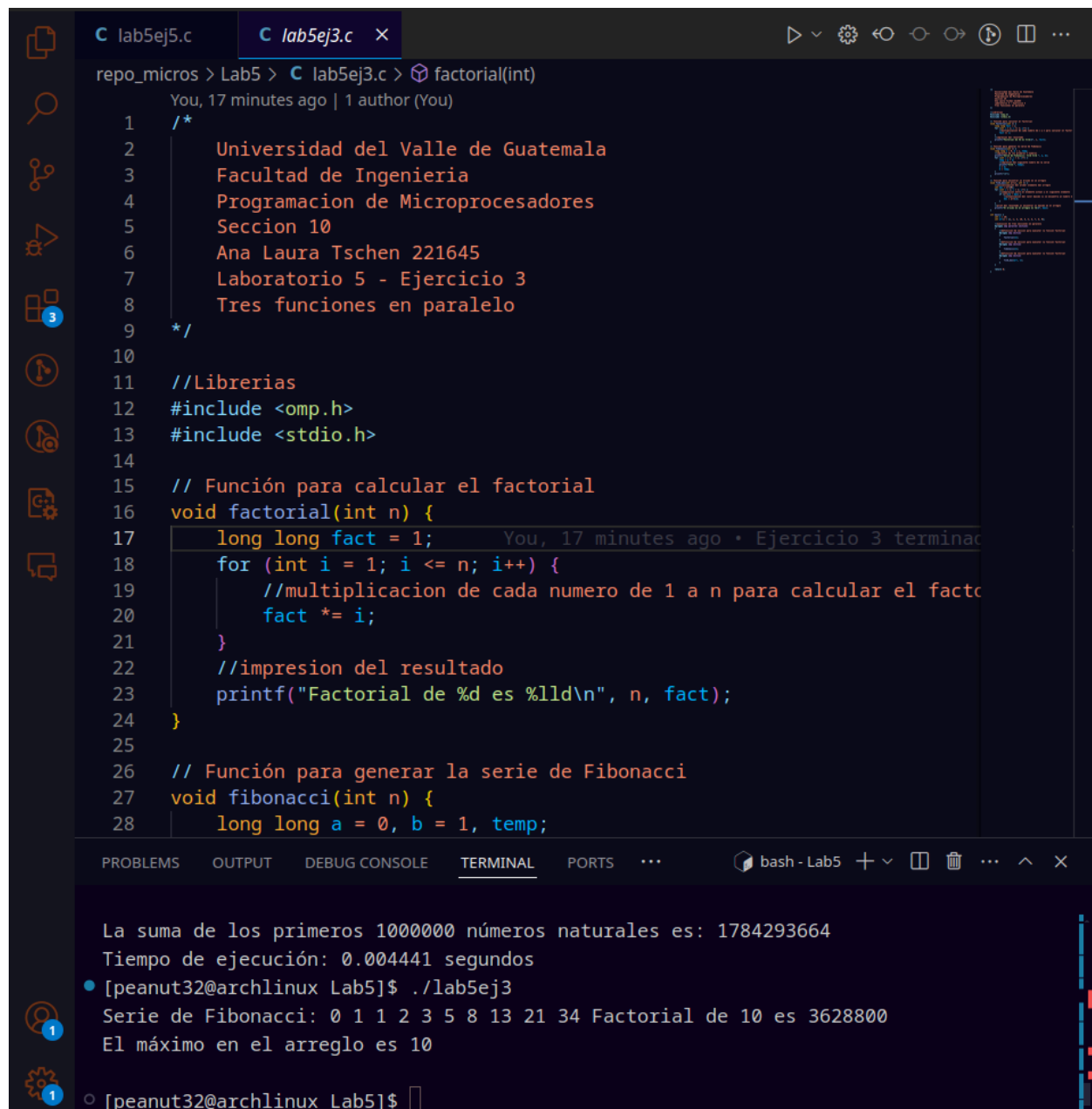
b) Usa `omp_get_wtime()` para medir los tiempos de ejecución.



```
repo_micros > Lab5 > C lab5ej2.c > main()
You, 25 minutes ago | 1 author (You)
1  /*
2     Universidad del Valle de Guatemala
3     Facultad de Ingenieria
4     Programacion de Microprocesadores
5     Seccion 10
6     Ana Laura Tschen 221645
7     Laboratorio 5 - Ejercicio 2
8     Suma de los primeros N numeros naturales con un ciclo for paralelo
9  */
10 //Librerias
11 #include <omp.h>
12 #include <stdio.h>
13
14 #define N 1000000
15
16 //funcion main
17 int main() {
18     //variable que almacena la suma de los numeros
19     int sum = 0;
20
21     //definicion de variable start time para calcular el tiempo de ejec
22     double start_time = omp_get_wtime();
23
24     //inicio de un ciclo for paralelo con reduction
25     #pragma omp parallel for reduction(+:sum)
26     for (int i = 1; i <= N; i++) {
27         sum += i;
28     }
29 }
```

```
[peanut32@archlinux repo_micros]$ cd Lab5
[peanut32@archlinux Lab5]$ ./lab5ej5
Numero de veces que 'key' aparece en paralelo: 3
[peanut32@archlinux Lab5]$ ./lab5ej2
La suma de los primeros 1000000 números naturales es: 1784293664
Tiempo de ejecución: 0.004441 segundos
[peanut32@archlinux Lab5]$
```

3. (15 pts.) Escribe un programa en C que ejecute tres funciones diferentes en paralelo usando la directiva `#pragma omp sections`. Cada sección debe ejecutar una función distinta, por ejemplo, una que calcule el factorial de un número, otra que genere la serie de Fibonacci, y otra que encuentre el máximo en un arreglo, operaciones matemáticas no simples. Asegúrate de que cada función sea independiente y no tenga dependencias con las otras.

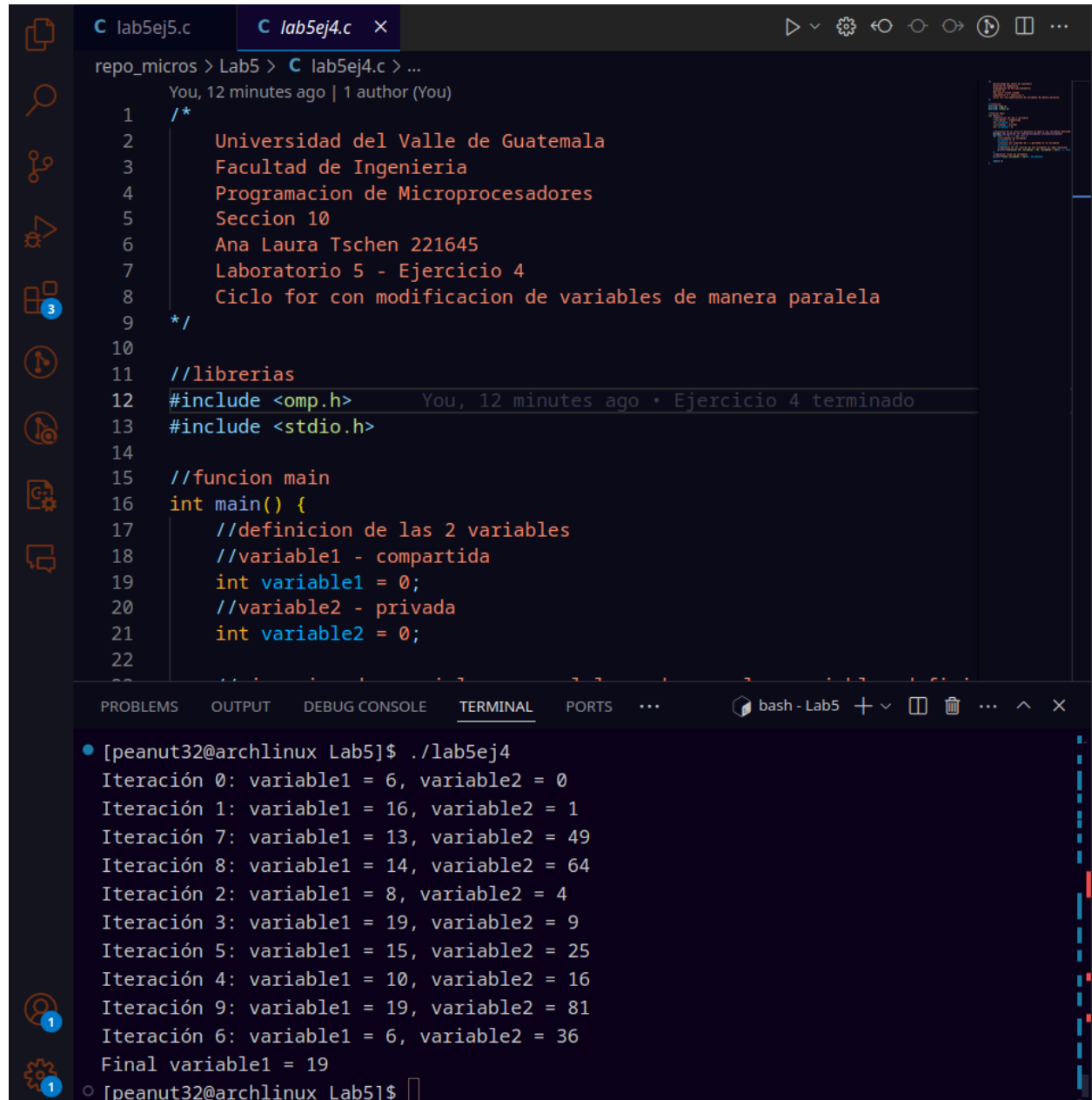


The screenshot shows a code editor with a C program and its execution output. The program is named `lab5ej3.c` and is located in the `Lab5` directory. The code includes comments identifying the author as Ana Laura Tschen 221645 and the course as Microprocesadores - Sección 10. The program defines three functions: `factorial`, `fibonacci`, and `maximo`. The `factorial` function calculates the factorial of a number `n` using a loop. The `fibonacci` function generates the first `n` numbers of the Fibonacci sequence. The `maximo` function finds the maximum value in an array. The program uses the `#pragma omp sections` directive to execute these three functions in parallel. The output shows the results of these functions: the sum of the first 1,000,000 natural numbers is 1784293664, the execution time is 0.004441 seconds, the Fibonacci sequence is 0 1 1 2 3 5 8 13 21 34, the factorial of 10 is 3628800, and the maximum value in the array is 10.

```
repo_micros > Lab5 > C lab5ej3.c > factorial(int)
You, 17 minutes ago | 1 author (You)
1  /*
2   Universidad del Valle de Guatemala
3   Facultad de Ingenieria
4   Programacion de Microprocesadores
5   Seccion 10
6   Ana Laura Tschen 221645
7   Laboratorio 5 - Ejercicio 3
8   Tres funciones en paralelo
9  */
10
11 //Librerias
12 #include <omp.h>
13 #include <stdio.h>
14
15 // Función para calcular el factorial
16 void factorial(int n) {
17     long long fact = 1;
18     for (int i = 1; i <= n; i++) {
19         //multiplicacion de cada numero de 1 a n para calcular el facto
20         fact *= i;
21     }
22     //impresion del resultado
23     printf("Factorial de %d es %lld\n", n, fact);
24 }
25
26 // Función para generar la serie de Fibonacci
27 void fibonacci(int n) {
28     long long a = 0, b = 1, temp;
```

La suma de los primeros 1000000 números naturales es: 1784293664
Tiempo de ejecución: 0.004441 segundos
[peanut32@archlinux Lab5]\$./lab5ej3
Serie de Fibonacci: 0 1 1 2 3 5 8 13 21 34 Factorial de 10 es 3628800
El máximo en el arreglo es 10
[peanut32@archlinux Lab5]\$

4. (15 pts.) Escribe un programa en C que tenga un ciclo for donde se modifiquen dos variables de manera paralela usando `#pragma omp parallel for`.
- Usa la cláusula `shared` para gestionar el acceso a la variable1 dentro del ciclo.
 - Usa la cláusula `private` para gestionar el acceso a la variable2 dentro del ciclo.
 - Prueba con ambas cláusulas y explica las diferencias observadas en los resultados.



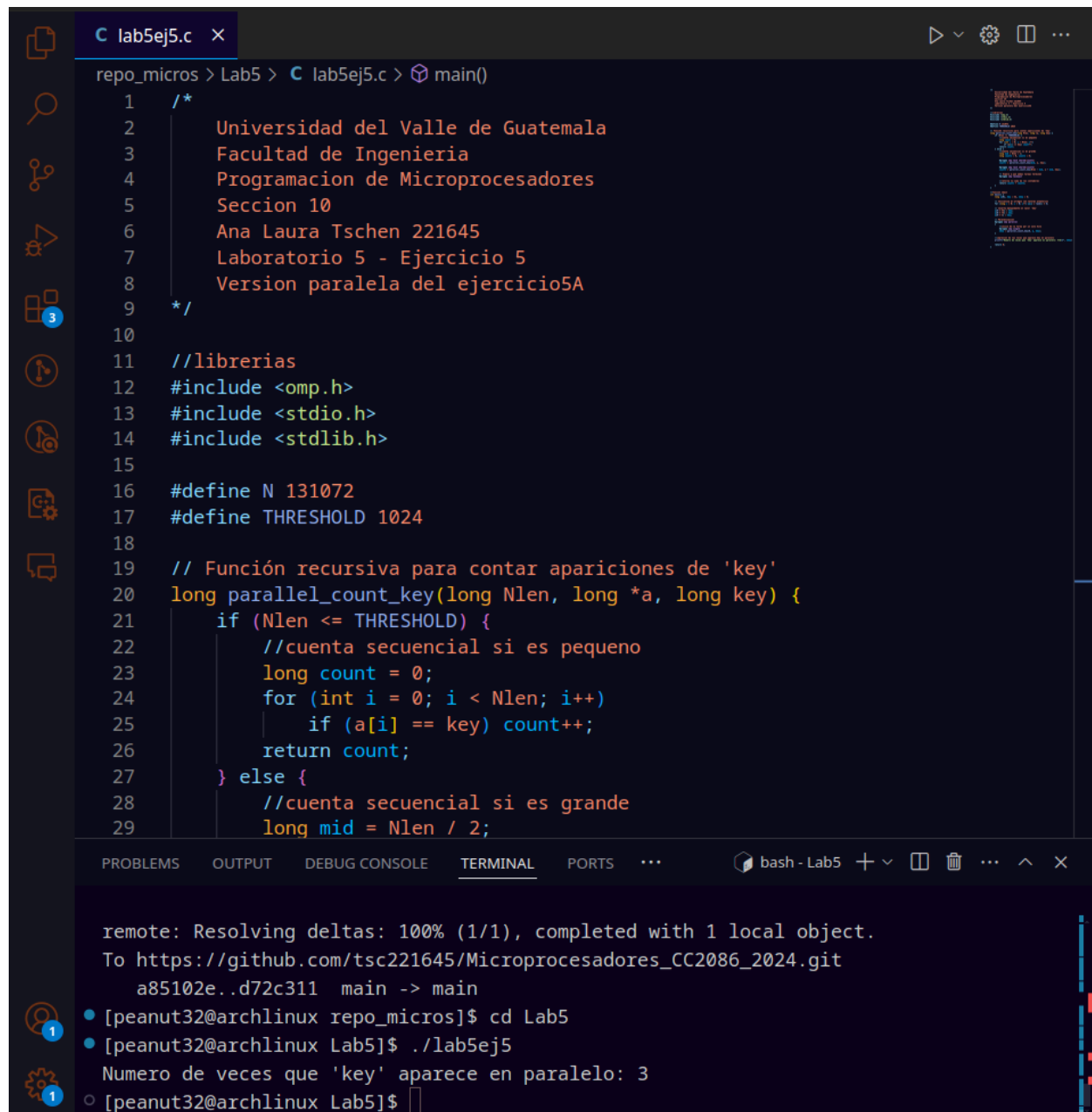
The image shows a code editor with a C program and a terminal window. The C program is a parallel for loop that updates two variables, `variable1` and `variable2`, in a loop from 0 to 10. `variable1` is shared and `variable2` is private. The terminal output shows the results of the program execution.

```
lab5ej5.c lab5ej4.c x
repo_micros > Lab5 > C lab5ej4.c > ...
You, 12 minutes ago | 1 author (You)
1  /*
2   Universidad del Valle de Guatemala
3   Facultad de Ingenieria
4   Programacion de Microprocesadores
5   Seccion 10
6   Ana Laura Tschen 221645
7   Laboratorio 5 - Ejercicio 4
8   Ciclo for con modificacion de variables de manera paralela
9  */
10
11 //librerias
12 #include <omp.h>
13 #include <stdio.h>
14
15 //funcion main
16 int main() {
17     //definicion de las 2 variables
18     //variable1 - compartida
19     int variable1 = 0;
20     //variable2 - privada
21     int variable2 = 0;
22
23     #pragma omp parallel for
24     for (int i = 0; i < 11; i++) {
25         variable1 = i * 2;
26         variable2 = i * 3;
27     }
28
29     printf("Final variable1 = %d\n", variable1);
30     return 0;
31 }
```

TERMINAL

```
bash - Lab5
[peanut32@archlinux Lab5]$ ./lab5ej4
Iteración 0: variable1 = 6, variable2 = 0
Iteración 1: variable1 = 16, variable2 = 1
Iteración 7: variable1 = 13, variable2 = 49
Iteración 8: variable1 = 14, variable2 = 64
Iteración 2: variable1 = 8, variable2 = 4
Iteración 3: variable1 = 19, variable2 = 9
Iteración 5: variable1 = 15, variable2 = 25
Iteración 4: variable1 = 10, variable2 = 16
Iteración 9: variable1 = 19, variable2 = 81
Iteración 6: variable1 = 6, variable2 = 36
Final variable1 = 19
[peanut32@archlinux Lab5]$
```

5. (30 pts.) Analiza el código en el programa Ejercicio_5A.c, que contiene un programa secuencial. Indica cuántas veces aparece un valor key en el vector a. Escribe una versión paralela en OpenMP utilizando una descomposición de tareas recursiva, en la cual se generen tantas tareas como hilos.



```
lab5ej5.c x
repo_micros > Lab5 > C lab5ej5.c > main()
1  /*
2     Universidad del Valle de Guatemala
3     Facultad de Ingenieria
4     Programacion de Microprocesadores
5     Seccion 10
6     Ana Laura Tschen 221645
7     Laboratorio 5 - Ejercicio 5
8     Version paralela del ejercicio5A
9  */
10
11 //librerias
12 #include <omp.h>
13 #include <stdio.h>
14 #include <stdlib.h>
15
16 #define N 131072
17 #define THRESHOLD 1024
18
19 // Función recursiva para contar apariciones de 'key'
20 long parallel_count_key(long Nlen, long *a, long key) {
21     if (Nlen <= THRESHOLD) {
22         //cuenta secuencial si es pequeno
23         long count = 0;
24         for (int i = 0; i < Nlen; i++)
25             if (a[i] == key) count++;
26         return count;
27     } else {
28         //cuenta secuencial si es grande
29         long mid = Nlen / 2;
30         long count1 = parallel_count_key(mid, a, key);
31         long count2 = parallel_count_key(Nlen - mid, a + mid, key);
32         return count1 + count2;
33     }
34 }
35
36 int main() {
37     long *a = (long *) malloc(N * sizeof(long));
38     long key = 100;
39     for (int i = 0; i < N; i++)
40         a[i] = (long) (rand() % 1000);
41     long count = parallel_count_key(N, a, key);
42     printf("Numero de veces que 'key' aparece en paralelo: %ld\n", count);
43     return 0;
44 }
```

remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/tsc221645/Microprocesadores_CC2086_2024.git
a85102e..d72c311 main -> main

```
• [peanut32@archlinux repo_micros]$ cd Lab5
• [peanut32@archlinux Lab5]$ ./lab5ej5
Numero de veces que 'key' aparece en paralelo: 3
○ [peanut32@archlinux Lab5]$
```

6. REFLEXIÓN DE LABORATORIO: se habilitará en una actividad independiente.