EXPERIMENT NO.1

Title: Study Basics of OpenMP API (Open Multi-Processor API)
Objective: To study the basics of OpenMP and demonstrate parallel programming using OpenMP.
Tool: GCC compiler with OpenMP support

🧠 Theory Recap

OpenMP (Open Multi-Processing) enables parallel programming for shared-memory architectures using simple compiler directives.
It allows developers to execute portions of code (like loops) concurrently across multiple processor cores.

💻 Executable Program

🔹 Program 1: Basic Parallel Region Example

```
// Filename: openmp_basic.c
#include <stdio.h>
#include <omp.h>

int main() {
   // Set number of threads (optional)
   omp_set_num_threads(4);

   // Parallel region begins
   #pragma omp parallel
   {
      int thread_id = omp_get_thread_num();   // get thread ID
      int total_threads = omp_get_num_threads(); // get total threads
      printf("Hello from thread %d out of %d threads.\n", thread_id, total_threads);
   }
   // Parallel region ends

   return 0;
}
```

🔹 Program 2: Parallel For Loop Example

```
// Filename: openmp_for_loop.c
#include <stdio.h>
#include <omp.h>

int main() {
   int i, n = 8;
   int a[8], b[8], c[8];

   // Initialize arrays
   for (i = 0; i < n; i++) {
      a[i] = i;
      b[i] = i * 2;
   }
```

```
    // Parallel for loop
    #pragma omp parallel for
    for (i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
        printf("Thread %d: c[%d] = %d\n", omp_get_thread_num(), i, c[i]);
    }

    printf("\nFinal Result (C array): ");
    for (i = 0; i < n; i++) {
        printf("%d ", c[i]);
    }
    printf("\n");

    return 0;
}
```

⚙️ Compilation & Execution

Step 1: Save the file

Example: openmp_basic.c or openmp_for_loop.c

Step 2: Compile with OpenMP flag

```
gcc -fopenmp openmp_basic.c -o openmp_basic
```

or

```
gcc -fopenmp openmp_for_loop.c -o openmp_for_loop
```

Step 3: Run the program

```
./openmp_basic
```

or

```
./openmp_for_loop
```

🧾 Expected Output

Output for openmp_basic.c

```
Hello from thread 0 out of 4 threads.
Hello from thread 1 out of 4 threads.
Hello from thread 2 out of 4 threads.
Hello from thread 3 out of 4 threads.
```

(Order may vary because threads execute concurrently.)

Output for openmp_for_loop.c

```
Thread 1: c[2] = 6
Thread 0: c[0] = 0
Thread 3: c[7] = 21
Thread 2: c[4] = 12
...
Final Result (C array): 0 3 6 9 12 15 18 21
```

🧩 Conclusion

OpenMP simplifies parallel programming by using compiler directives (#pragma omp).

The example demonstrates:

- How to create parallel regions
- How to use multiple threads to execute parts of the code simultaneously
- How to perform data-parallel operations efficiently using #pragma omp parallel for