**[ SUPRATIM NAG/CSE-AIML/22/57 ]**

**Open MP Code Assignment :**

1. **Write an Openmp Code to find out the absolute value of PI :**

Source Code:

#include <stdio.h>

#include <omp.h>

int main() {

double pi;

#pragma omp parallel

{

#pragma omp single

{

pi = 22.0 / 7.0;

printf("The value of Pi using 22/7 approximation is: %f\n", pi);

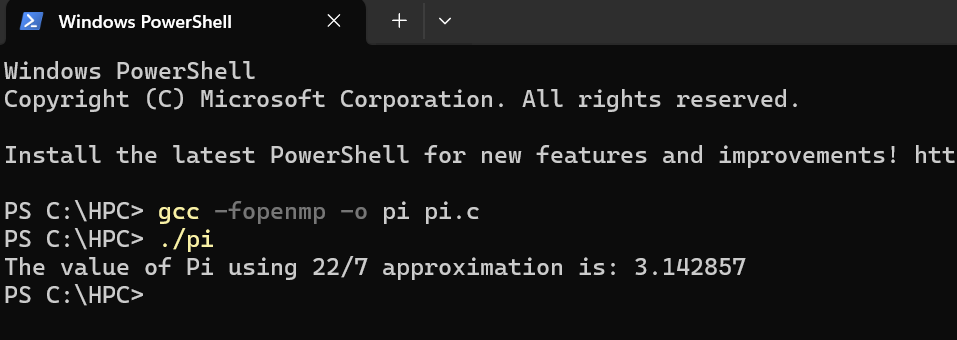
}

}

return 0;

}

**Output:**

****

1. **Write an Openmp code to generate the fibonacci number from 1-100**

Source Code:

#include <stdio.h>

#include <omp.h>

#define MAX\_FIB 100

void generate\_fibonacci(int \*fib\_sequence, int n) {

if (n >= 1) fib\_sequence[0] = 0; // First Fibonacci number

if (n >= 2) fib\_sequence[1] = 1; // Second Fibonacci number

// Declare loop variable outside of the for loop

int i;

#pragma omp parallel for

for (i = 2; i < n; i++) {

fib\_sequence[i] = fib\_sequence[i - 1] + fib\_sequence[i - 2];

}

}

int main() {

int fib\_sequence[MAX\_FIB];

// Generate Fibonacci numbers up to the largest number less than or equal to 100

generate\_fibonacci(fib\_sequence, MAX\_FIB);

int j;

printf("Fibonacci numbers from 1 to 100:\n");

for (j = 0; j < MAX\_FIB; j++) {

if (fib\_sequence[j] > 100) break;

printf("%d ", fib\_sequence[j]);

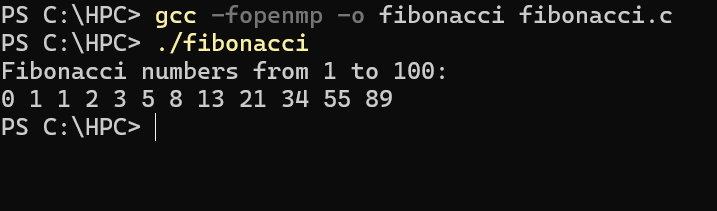
}

printf("\n");

return 0;

}

**Output:**

****

1. **Write an Openmp code to perform 2D matrix multiplication :**

Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 100 // Define the size of the matrices

void matrix\_multiply(int a[N][N], int b[N][N], int c[N][N]) {

int i, j, k;

#pragma omp parallel for private(j, k)

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

c[i][j] = 0; // Initialize the result cell

for (k = 0; k < N; k++) {

c[i][j] += a[i][k] \* b[k][j]; // Perform multiplication and addition

}

}

}

}

int main() {

int a[N][N], b[N][N], c[N][N];

int i,j;

// Initialize matrices a and b with random values

for (i = 0; i < N; i++) {

for ( j = 0; j < N; j++) {

a[i][j] = rand() % 10; // Random values between 0 and 9

b[i][j] = rand() % 10; // Random values between 0 and 9

}

}

// Perform matrix multiplication

matrix\_multiply(a, b, c);

// Print the resulting matrix c

printf("Resulting Matrix C after multiplication:\n");

for (i = 0; i < N; i++) {

for ( j = 0; j < N; j++) {

printf("%d ", c[i][j]);

}

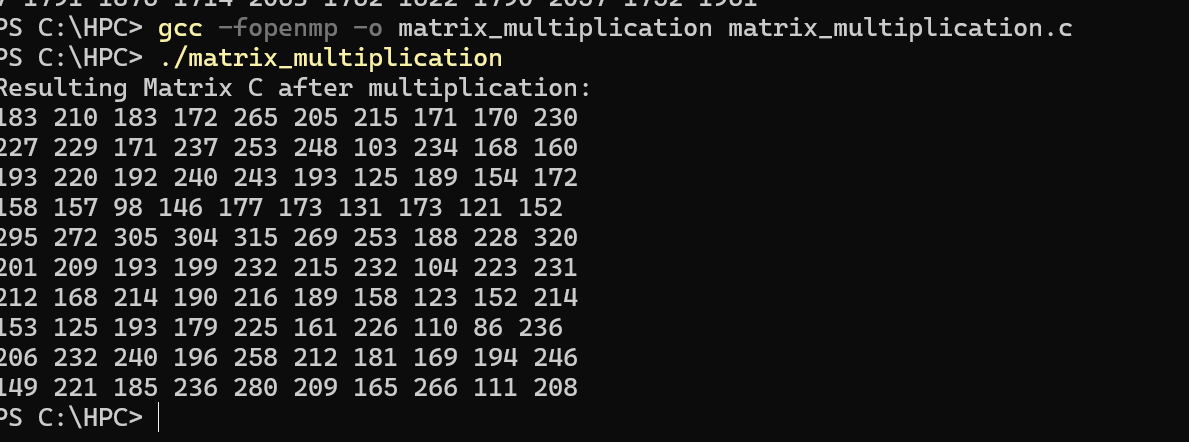
printf("\n");

}

return 0;

}

**Output :**

****

1. **Write an Openmp code to implement the factorial of a number using recursion :**

Source Code:

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

// Function to calculate factorial recursively

long long factorial(int n) {

if (n == 0) // Base case: 0! = 1

return 1;

else

return n \* factorial(n - 1); // Recursive call

}

int main(int argc, char \*argv[]) {

if (argc != 2) { // Check for the correct number of arguments

printf("Usage: %s <non-negative integer>\n", argv[0]);

return 1;

}

int number = atoi(argv[1]); // Convert argument to integer

if (number < 0) { // Check if the number is negative

printf("Factorial is not defined for negative numbers.\n");

return 1;

}

long long result;

// Parallel region for calculating factorial

#pragma omp parallel

{

#pragma omp single // Only one thread executes this block

{

result = factorial(number); // Calculate factorial

}

}

printf("Factorial of %d is %lld\n", number, result); // Print result

return 0;

}

// Parallel region for calculating factorial

#pragma omp parallel

{

#pragma omp single // Only one thread executes this block

{

result = factorial(number); // Calculate factorial

}

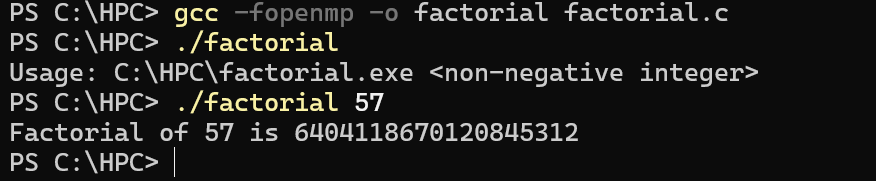
}

printf("Factorial of %d is %lld\n", number, result); // Print result

return 0;

}

**Output :**

****