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

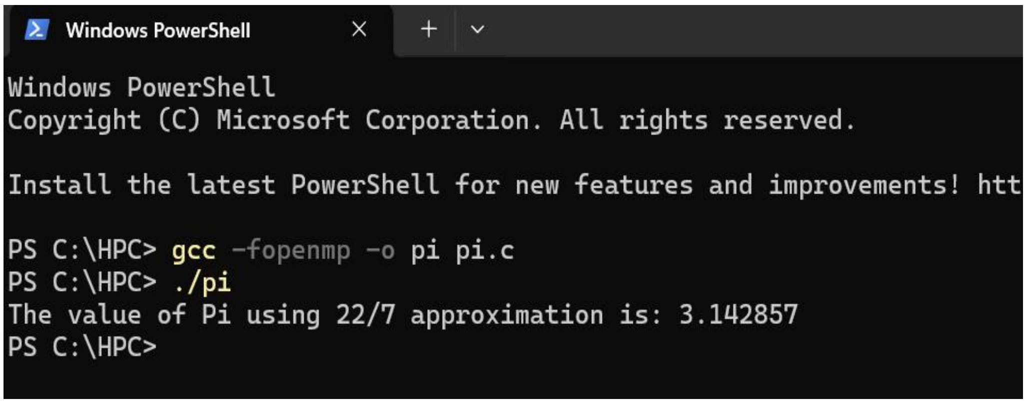
Open MP Code Assignment :

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:



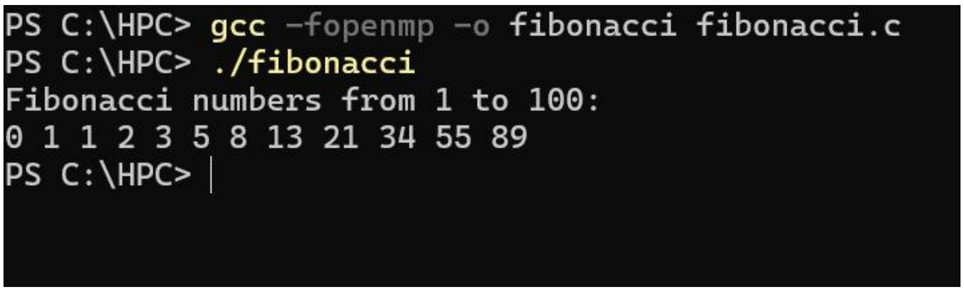
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! htt

PS C:\HPC> gcc -fopenmp -o pi pi.c
PS C:\HPC> ./pi
The value of Pi using 22/7 approximation is: 3.142857
PS C:\HPC>
```

2) 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:



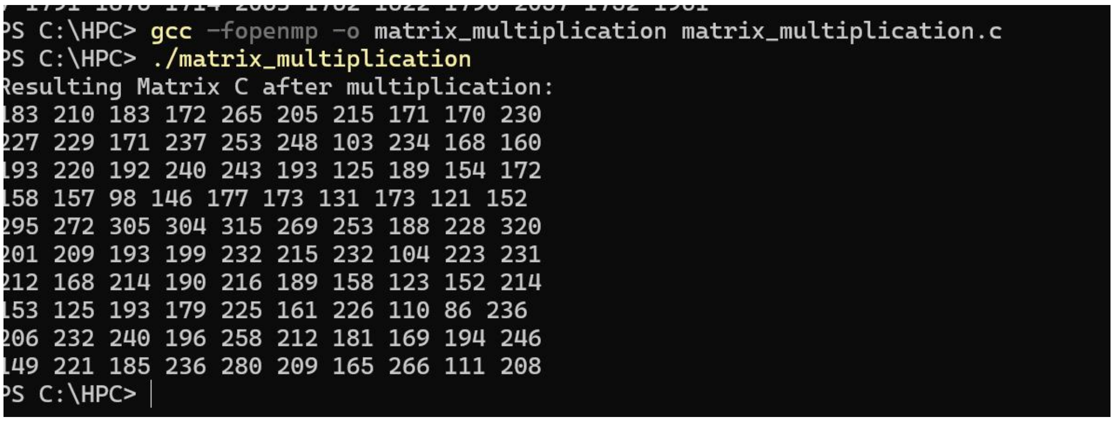
```
PS C:\HPC> gcc -fopenmp -o fibonacci fibonacci.c
PS C:\HPC> ./fibonacci
Fibonacci numbers from 1 to 100:
0 1 1 2 3 5 8 13 21 34 55 89
PS C:\HPC>
```

3) 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 :



```
PS C:\HPC> gcc -fopenmp -o matrix_multiplication matrix_multiplication.c
PS C:\HPC> ./matrix_multiplication
Resulting Matrix C after multiplication:
183 210 183 172 265 205 215 171 170 230
227 229 171 237 253 248 103 234 168 160
193 220 192 240 243 193 125 189 154 172
158 157 98 146 177 173 131 173 121 152
295 272 305 304 315 269 253 188 228 320
201 209 193 199 232 215 232 104 223 231
212 168 214 190 216 189 158 123 152 214
153 125 193 179 225 161 226 110 86 236
206 232 240 196 258 212 181 169 194 246
149 221 185 236 280 209 165 266 111 208
PS C:\HPC>
```

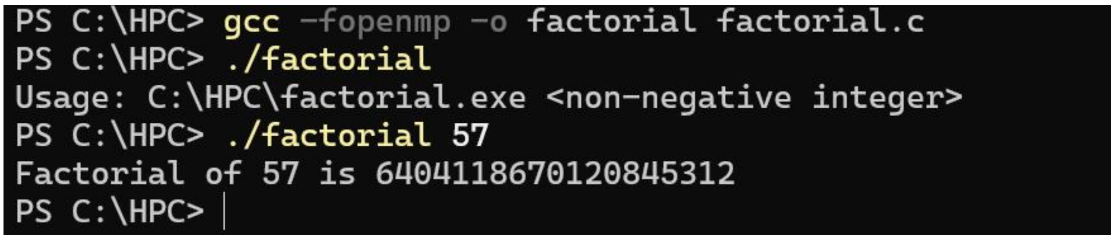
4) Write an Openmp code to implement the factorial of a number using recursion :

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;
}
```

Output :



```
PS C:\HPC> gcc -fopenmp -o factorial factorial.c
PS C:\HPC> ./factorial
Usage: C:\HPC\factorial.exe <non-negative integer>
PS C:\HPC> ./factorial 57
Factorial of 57 is 6404118670120845312
PS C:\HPC>
```