Exercise 1 : OpenMP

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int A[100];
    srand(1234);
    for(i=0;i<100;i++)
    {
        A[100] = rand()%1000;
    }
}
```

สิ่งที่ต้องใช้

```
// Ubuntu ที่ติดตั้ง GCC และ omp เรียบร้อยแล้ว
#include <omp.h> // ไลบรารี่ OpenMP
#pragma omp parallel // สำหรับการแบ่งงานให้แต่ละ Thread
```

Pragma คืออะไร?

Pragma เป็นชุดคำสั่งชนิดหนึ่งสามารถเปิดปิดฟังก์ชั่นบางอย่างได้ เช่น pragma omp การใช้ pragma omp คือการอนุญาตให้โค้ดนั้นสามารถคอมไพล์ (Complie) ลงถึงระดับ OS (Operating System) ในรูปแบบขนาน(Parallel)

นี่คือโจทย์ที่เราจะต้องแก้ให้ตรงตามเงื่อนไขของอาจารย์

ข้อ 1.

Given an integer array A[100], write an OpenMP program that multiples 10 to each element of the array.
Verify your output with the sequential version.

เราจะต้องใช้ OpenMP เข้ามาเพื่อคูณค่าแต่ละตัวลงอาเรย์

```
#include <omp.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
int main()
{
    int A[100];
    srand(1234);
    #pragma omp parallel for // เติม for สำหรับการรัน for loop แบ่งงานให้แต่ละ Thread
    for(i=0;i<100;i++)
    {
        A[i] = rand()%1000;
        printf("%d",A[i]*10);
    }
    return 0;
}
```

วิธีการคอมไพล์และรันไฟล์

วิธีการคอมไพล์โค้ดที่เขียนด้วย OpenMP มีวิธีดังนี้ (วิธีนี้ใช้บน Ubuntu)

```
1. เปิด Terminal ในที่ที่เราเก็บโค้ดไว้
2. พิมพ์คำสั่งนี้ลงไปแล้วกด Enter


gcc -o ชื่อไฟล์ที่จะเอาไว้รัน -fopenmp ชื่อไฟล์ที่เราสร้าง.c
# ตัวอย่าง
gcc -o openmp5 -fopenmp openmp5.c


3. เมื่อ Enter แล้วถ้าไม่มี Error ให้พิมพ์คำสั่งต่อไปนี้เพื่อรัน


./openmp5 # เป็นต้น
```

ข้อ 2.

Write an OpenMP program to find the summation of values in A[] using reduction clause (+operator).
Verify your output with the sequential version.

เราจะต้องใช้ reduction clause เพื่อลดรูปเครื่องหมายสมการ

```c
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  int a[100],sum;
  srand(1234);
    // Beginning of parallel region
    #pragma omp parallel for // pragma สำหรับ for สำหรับการรัน for loop
        for (int i = 0; i < 100; i++)
        {
            a[i] = rand() %1000;
            // a[i] = i;
            /* code */
            sum += a[i]; // Reduction clause instead
        }
        printf("%d\n",sum);
    return 0;
    // Ending of parallel region
}
```

ตรง sum += a[i] คือการลดรูปจาก sum = sum + a[i] หรือที่เรียกว่า Reduction Clause ทำให้สมการนั้นดูสั้นขึ้น (แต่จะแอบเข้าใจยากอยู่นิดหน่อย) และอ่านได้ง่ายขึ้น

ข้อ 3.

Write an OpenMP program to find the maximum values in an integer array using reduction clause (max operator).
Verify your output with the sequential version.

เราจะต้องสร้าง int max ขึ้นมา 1 ตัว แล้วสร้าง if condition เพื่อหาค่าที่สูงที่สุดใน array นั้นๆ

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  int a[100],max; // สร้าง int ชื่อ max มาเพื่อเก็บค่าที่สูงที่สุด
  srand(1234);
    // Beginning of parallel region
    #pragma omp parallel for
        for (int i = 0; i < 100; i++)
        {
            a[i] = rand() %1000;
            if(a[i]>max) // ถ้าในอาเรย์มีค่าสูงกว่า max
            {
                max = a[i]; // ค่า max จะเท่ากับ array ตัวนั้นๆ
            }
            /* code */
        }
        printf("%d\n",max);
    return 0;
    // Ending of parallel region
}
```

ข้อ 4.

Write an OpenMP program to find the number of integer greater or equal to 500.
Verify your output with the sequential version.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  int a[100],max;
  srand(1234);
    // Beginning of parallel region
    #pragma omp parallel for
        for (int i = 0; i < 100; i++)
        {
            a[i] = rand() % 1000;
            if(a[i]>=500)
            {
                printf("%d \n",a[i]);
            }
            /* code */
        }
    return 0;
    // Ending of parallel region
}
```

ข้อ 5.

Rewrite question 2 (find summation) by not using reduction clause.
Use private variable (psum) and shared variable (sum) with critical section.

เราจะต้องเอาข้อ 2. มาแก้ใหม่โดยไม่ใช้การลดรูปเครื่องหมายสมการ แต่ให้ใช้ตัวแปรแบบ private และ shared พร้อมทั้งใช้ Critical

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
  int a[100],i,max=0,sum=0,psum=0;
  srand(1234);
    // Beginning of parallel region
    #pragma omp parallel default(none) private(psum,i) shared(sum,a) num_threads(3)
    /**
     * ประกาศค่า default ว่าไม่ใช้
     * ประกาศ private ว่าใช้ตัวแปรใดได้บ้าง
     * ประกาศ shared ว่าใช้ร่วมกับตัวใดได้บ้าง
     * และประกาศการใช้ num_threads 3 ตัว
     * **/
    {
        for (int i = 0; i < 100; i++)
        {
            a[i] = rand() %1000;
            psum += a[i];
            // a[i] = i;
            /* code */
        }
        // Go to the Critical sections like a critical region (Mutex)
        #pragma omp critical // เข้า critical Zone หรือ Critical Region
        sum += psum;
    }
    printf("%d\n",sum);
    // Ending of parallel region
}
```