## **CAO Lab Assignment 1**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Problem statement 1:**

BMI and find Thread number and master thread in c using OMP function.

#include <stdio.h>

#include <omp.h>

int main() {

double weight, height, bmi;

// Take weight and height as input for one person (serial)

printf("Enter weight (kg): ");

scanf("%lf", &weight);

printf("Enter height (m): ");

scanf("%lf", &height);

int num\_procs = omp\_get\_num\_procs();

printf("Total processors: %d\n", num\_procs);

#pragma omp parallel

{

int thread\_id = omp\_get\_thread\_num();

#pragma omp master

{

int num\_threads = omp\_get\_num\_threads();

printf("Total threads: %d\n", num\_threads);

}

if (thread\_id == 0) {

printf("Thread %d: This is the master thread.\n", thread\_id);

} else {

printf("Thread %d: This is not the master thread.\n", thread\_id);

}

#pragma omp single

{

bmi = weight / (height \* height);

}

#pragma omp barrier

if (thread\_id == 0) {

printf("Weight = %.2f kg, Height = %.2f m, BMI = %.2f\n",

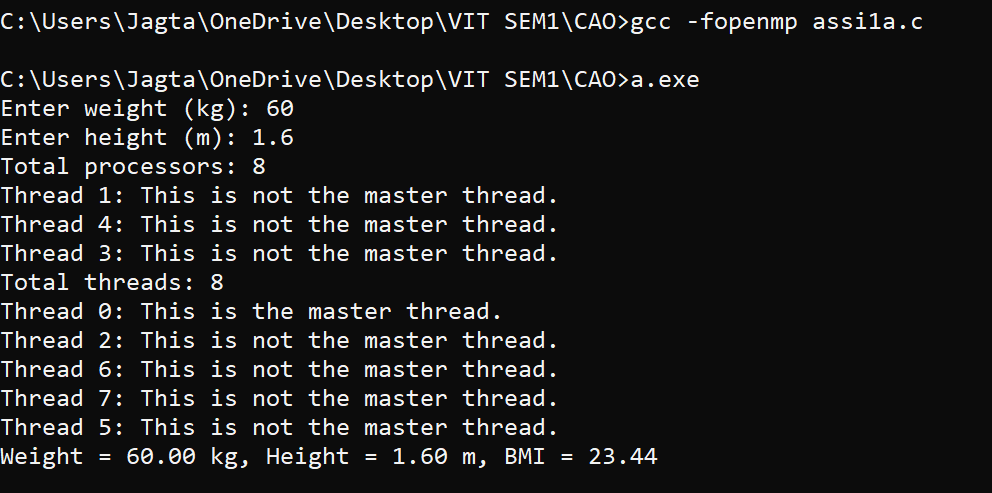
weight, height, bmi);

}

}

return 0;

}



**Problem statement 2:**

Find sum of 2 arrays and print the result in the third array and find Thread number and master thread in c using OMP function.

#include <stdio.h>

#include <omp.h>

#define MAX\_SIZE 100

int main() {

int i, size1, size2;

printf("Enter the size of array 1 (max %d): ", MAX\_SIZE);

scanf("%d", &size1);

printf("Enter the size of array 2 (max %d): ", MAX\_SIZE);

scanf("%d", &size2);

int array1[MAX\_SIZE] = {0}, array2[MAX\_SIZE] = {0}, result[MAX\_SIZE] = {0};

printf("Enter elements for array 1:\n");

for (i = 0; i < size1; i++) {

printf("Element %d: ", i);

scanf("%d", &array1[i]);

}

printf("Enter elements for array 2:\n");

for (i = 0; i < size2; i++) {

printf("Element %d: ", i);

scanf("%d", &array2[i]);

}

#pragma omp parallel

{

int thread\_id = omp\_get\_thread\_num();

#pragma omp master

{

int num\_threads = omp\_get\_num\_threads();

printf("Total threads: %d\n", num\_threads);

int num\_procs = omp\_get\_num\_procs();

printf("Total processors: %d\n", num\_procs);

}

if (thread\_id == 0) {

printf("Thread %d: This is the master thread.\n", thread\_id);

} else {

printf("Thread %d: This is not the master thread.\n", thread\_id);

}

int min\_size = (size1 < size2) ? size1 : size2;

#pragma omp for

for (i = 0; i < min\_size; i++) {

result[i] = array1[i] + array2[i];

}

#pragma omp single

{

if (size1 > size2) {

for (i = size2; i < size1; i++) {

result[i] = array1[i];

}

} else if (size2 > size1) {

for (i = size1; i < size2; i++) {

result[i] = array2[i];

}

}

}

#pragma omp barrier

#pragma omp single

{

int max\_size = (size1 > size2) ? size1 : size2;

printf("Resulting array:\n");

for (i = 0; i < max\_size; i++) {

printf("Element %d: %d\n", i, result[i]);

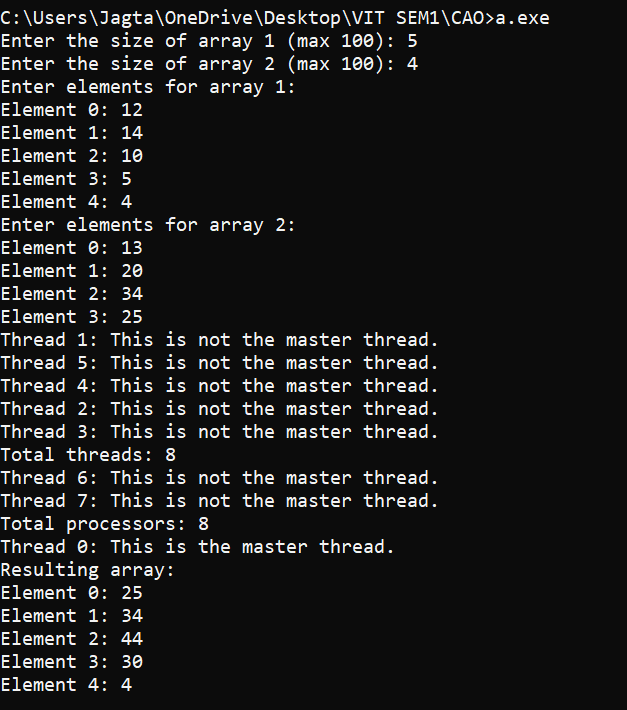
}

}

}

return 0;

}



**Problem statement 3:**

Print sum of odd numbers and even numbers in an array and find Thread number and master thread in c using OMP function.

#include <stdio.h>

#include <omp.h>

#define MAX\_SIZE 100

int main() {

int i, size;

int array[MAX\_SIZE];

int sum\_even = 0, sum\_odd = 0;

printf("Enter the size of the array (max %d): ", MAX\_SIZE);

scanf("%d", &size);

if (size > MAX\_SIZE) {

printf("Size exceeds maximum allowed value of %d.\n", MAX\_SIZE);

return 1;

}

printf("Enter elements of the array:\n");

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

printf("Element %d: ", i);

scanf("%d", &array[i]);

}

#pragma omp parallel

{

// Get the thread ID

int thread\_id = omp\_get\_thread\_num();

int num\_threads = omp\_get\_num\_threads();

// Print total threads information only once

#pragma omp master

{

printf("Total threads: %d\n", num\_threads);

}

// Print thread-specific information

printf("Thread %d: ", thread\_id);

if (thread\_id == 0) {

printf("This is the master thread.\n");

} else {

printf("This is not the master thread.\n");

}

// Private variables for sums

int local\_sum\_even = 0;

int local\_sum\_odd = 0;

#pragma omp for

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

if (array[i] % 2 == 0) {

local\_sum\_even += array[i];

} else {

local\_sum\_odd += array[i];

}

}

#pragma omp critical

{

sum\_even += local\_sum\_even;

sum\_odd += local\_sum\_odd;

}

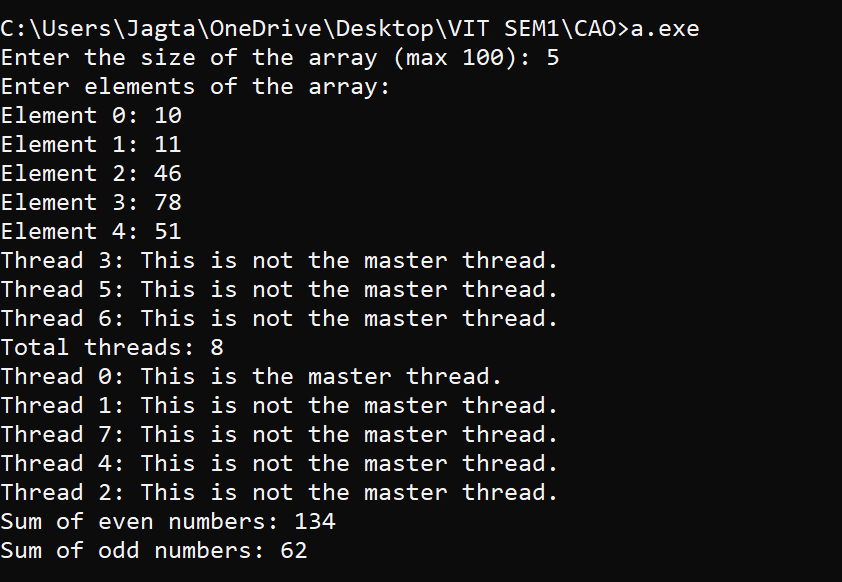
}

printf("Sum of even numbers: %d\n", sum\_even);

printf("Sum of odd numbers: %d\n", sum\_odd);

return 0;

}



## **CAO Lab Assignment 2**

**Name: Mahesh Jagtap**

**Reg no. 24MCS1017**

**Problem statement 1:**

The election commission has decided to organise a special camp to include young people(age greater than or equal to 16 and less than 18) in electoral role. Help the officials to identify the eligible people. Use thread “1” to print eligible people and thread “0” to not eligible candidate. Get minimum 10 people data.

#include <stdio.h>

#include <omp.h>

#define MAX\_PEOPLE 100 // Define a maximum number of people

int main() {

int ages[MAX\_PEOPLE]; // Array to store ages

int eligible[MAX\_PEOPLE]; // Array to store eligibility (1 for eligible, 0 for not eligible)

int N; // Number of people

// Get the number of people from the user

printf("Enter the number of people: ");

scanf("%d", &N);

// Ensure N does not exceed the maximum limit

if (N > MAX\_PEOPLE) {

printf("Number of people exceeds maximum limit of %d.\n", MAX\_PEOPLE);

return 1;

}

// Get the ages from the user

printf("Enter the ages of the people:\n");

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

scanf("%d", &ages[i]);

}

// Determine eligibility in the main thread

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

eligible[i] = (ages[i] >= 16 && ages[i] < 18) ? 1 : 0;

}

// Start parallel region with two threads for printing

#pragma omp parallel num\_threads(2)

{

int thread\_num = omp\_get\_thread\_num();

// Thread 0: Print non-eligible people

if (thread\_num == 0) {

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

if (eligible[i] == 0) {

printf("Thread %d: Person with age %d is not eligible.\n", thread\_num, ages[i]);

}

}

}

// Thread 1: Print eligible people

if (thread\_num == 1) {

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

if (eligible[i] == 1) {

printf("Thread %d: Person with age %d is eligible.\n", thread\_num, ages[i]);

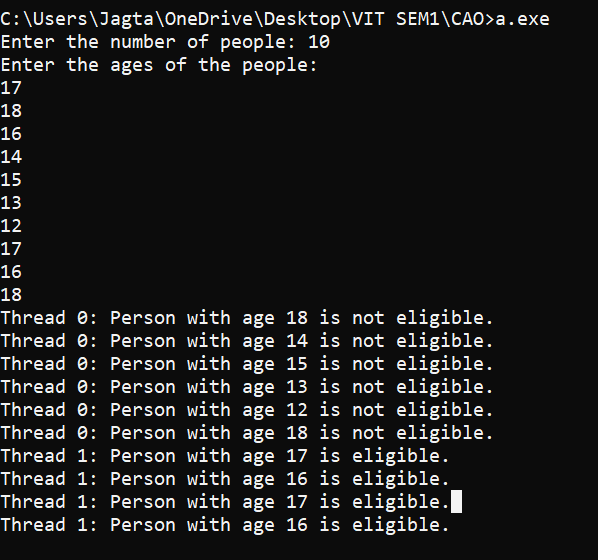
}

}

} }

return 0;

}



**Problem statement 2:**

For the above election commission using section calculate

i. The total no. of eligible candidates and

ii. Total no. of not eligible candidates

#include <stdio.h>

#include <omp.h>

#define MAX\_PEOPLE 100 // Define a maximum number of people

int main() {

int ages[MAX\_PEOPLE]; // Array to store ages

int eligible[MAX\_PEOPLE]; // Array to store eligibility (1 for eligible, 0 for not eligible)

int N; // Number of people

// Get the number of people from the user

printf("Enter the number of people: ");

scanf("%d", &N);

// Ensure N does not exceed the maximum limit

if (N > MAX\_PEOPLE) {

printf("Number of people exceeds maximum limit of %d.\n", MAX\_PEOPLE);

return 1;

}

// Get the ages from the user

printf("Enter the ages of the people:\n");

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

scanf("%d", &ages[i]);

}

// Determine eligibility

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

eligible[i] = (ages[i] >= 16 && ages[i] < 18) ? 1 : 0;

}

// Initialize counters

int eligible\_count = 0;

int not\_eligible\_count = 0;

// Start parallel region with sections for counting

#pragma omp parallel sections

{

#pragma omp section

{

// Count non-eligible people

int local\_not\_eligible\_count = 0;

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

if (eligible[i] == 0) {

local\_not\_eligible\_count++;

}

}

#pragma omp atomic

not\_eligible\_count += local\_not\_eligible\_count;

}

#pragma omp section

{

// Count eligible people

int local\_eligible\_count = 0;

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

if (eligible[i] == 1) {

local\_eligible\_count++;

}

}

#pragma omp atomic

eligible\_count += local\_eligible\_count;

}

}

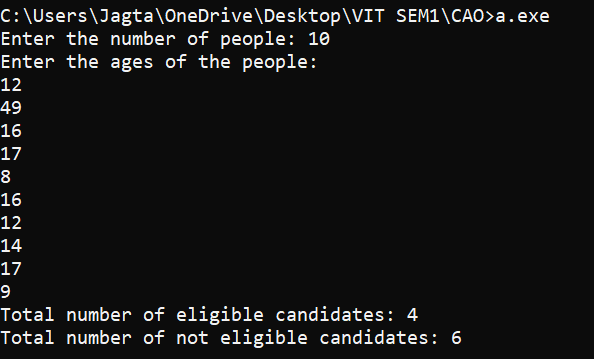
// Print the results

printf("Total number of eligible candidates: %d\n", eligible\_count);

printf("Total number of not eligible candidates: %d\n", not\_eligible\_count);

return 0;

}



## **CAO Lab Assignment 3**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Problem statement:**

Consider you have to write a program for the VIT placement cell where 10 students are placed in 4 companies namely, Amazon, Google, Shell, and Intel. Assume no student is offered more than one placement offer. The program has to do the following tasks in parallel and display the result with thread id. Use separate sections to perform each operation

•Get as input the name, register number, the pay package of students selected for jobs in the particular organization

•Display the total number of students selected in each company.

•Display the average pay package of the 10 students

Calculate the execution time of each of the above processes using wtime.

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <string.h>

#define NUM\_STUDENTS 10

// Define a structure for student

typedef struct {

char name[50];

int reg\_no;

float pay\_package;

char company[50];

} Student;

// Array to hold student data

Student students[NUM\_STUDENTS];

// Function to get student data

void get\_student\_data() {

for (int i = 0; i < NUM\_STUDENTS; i++) {

printf("Enter details for student %d\n", i + 1);

printf("Name: ");

scanf("%s", students[i].name);

printf("Register Number: ");

scanf("%d", &students[i].reg\_no);

printf("Pay Package: ");

scanf("%f", &students[i].pay\_package);

printf("Company (Amazon/Google/Shell/Intel): ");

scanf("%s", students[i].company);

}

}

// Function to display total students per company

void display\_total\_students\_per\_company() {

int count\_amazon = 0, count\_google = 0, count\_shell = 0, count\_intel = 0;

#pragma omp parallel

{

#pragma omp for

for (int i = 0; i < NUM\_STUDENTS; i++) {

if (strcmp(students[i].company, "Amazon") == 0) {

#pragma omp atomic

count\_amazon++;

} else if (strcmp(students[i].company, "Google") == 0) {

#pragma omp atomic

count\_google++;

} else if (strcmp(students[i].company, "Shell") == 0) {

#pragma omp atomic

count\_shell++;

} else if (strcmp(students[i].company, "Intel") == 0) {

#pragma omp atomic

count\_intel++;

}

}

}

printf("Total students per company:\n");

printf("Amazon: %d\n", count\_amazon);

printf("Google: %d\n", count\_google);

printf("Shell: %d\n", count\_shell);

printf("Intel: %d\n", count\_intel);

}

// Function to display average pay package

void display\_average\_pay\_package() {

float total\_pay = 0.0;

#pragma omp parallel

{

#pragma omp for reduction(+:total\_pay)

for (int i = 0; i < NUM\_STUDENTS; i++) {

total\_pay += students[i].pay\_package;

}

}

float average\_pay = total\_pay / NUM\_STUDENTS;

printf("Average pay package: %.2f\n", average\_pay);

}

int main() {

double start\_time, end\_time;

start\_time = omp\_get\_wtime();

get\_student\_data();

end\_time = omp\_get\_wtime();

printf("Time taken for input student data: %f seconds\n", end\_time - start\_time);

// Parallel region with sections

#pragma omp parallel sections

{

#pragma omp section

{

double section\_start\_time = omp\_get\_wtime();

display\_total\_students\_per\_company();

double section\_end\_time = omp\_get\_wtime();

printf("Time taken for displaying total students per company: %f seconds\n", section\_end\_time - section\_start\_time);

}

#pragma omp section

{

double section\_start\_time = omp\_get\_wtime();

display\_average\_pay\_package();

double section\_end\_time = omp\_get\_wtime();

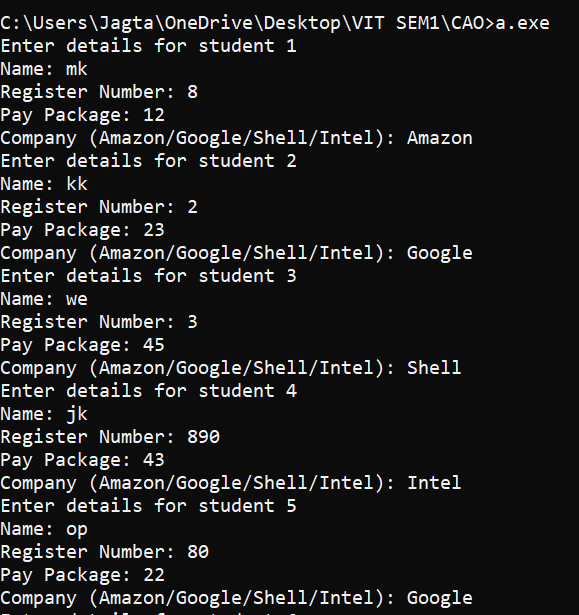
printf("Time taken for calculating average pay package: %f seconds\n", section\_end\_time - section\_start\_time);

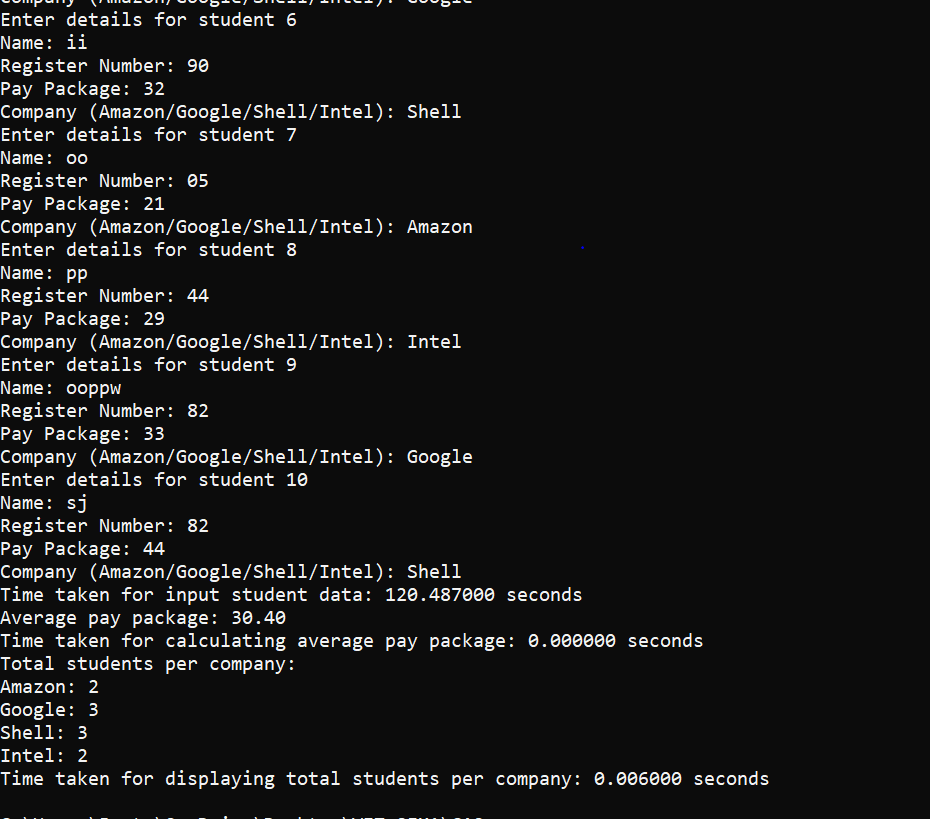
}

}

return 0;

}





## **CAO Lab Assignment 4**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

Problem statement: A new aeroplane service company announces a new scheme for ticket reservation as its opening ceremony offers. It runs a 100-seat plane with 3 services a day. The booking scheme is as follows:

* For the first 20 passengers 40% of the original ticket cost;
* For the next 20 passengers, it provides a 30% discount on the original ticket cost;
* For the next 30 passengers it gives 25% off the original ticket cost;
* For the remaining passengers, it gives a 10% off the original ticket cost. Assume that all the tickets are sold for each service.

Write an OpenMP C program to calculate the amount earned by the company in a month. Identify the master thread, display the processor number, and use sections appropriately. For each loop calculate its wall time.

#include <stdio.h>

#include <omp.h>

#define ORIGINAL\_TICKET\_COST 1000 // Assume the original ticket cost is 1000

#define SEATS 100

#define SERVICES\_PER\_DAY 3

#define DAYS\_IN\_MONTH 30

int main() {

int i, total\_earnings = 0;

double start\_time, end\_time;

omp\_set\_num\_threads(4); // Set the number of threads

start\_time = omp\_get\_wtime();

#pragma omp parallel sections

{

#pragma omp section

{

int earnings = 0;

for (i = 0; i < 20; i++) {

earnings += ORIGINAL\_TICKET\_COST \* 0.6; // 40% off

}

#pragma omp critical

total\_earnings += earnings;

printf("Section 1: Master thread: %d, Processor: %d, Earnings: %d\n", omp\_get\_thread\_num(), omp\_get\_num\_procs(), earnings);

}

#pragma omp section

{

int earnings = 0;

for (i = 20; i < 40; i++) {

earnings += ORIGINAL\_TICKET\_COST \* 0.7; // 30% off

}

#pragma omp critical

total\_earnings += earnings;

printf("Section 2: Master thread: %d, Processor: %d, Earnings: %d\n", omp\_get\_thread\_num(), omp\_get\_num\_procs(), earnings);

}

#pragma omp section

{

int earnings = 0;

for (i = 40; i < 70; i++) {

earnings += ORIGINAL\_TICKET\_COST \* 0.75; // 25% off

}

#pragma omp critical

total\_earnings += earnings;

printf("Section 3: Master thread: %d, Processor: %d, Earnings: %d\n", omp\_get\_thread\_num(), omp\_get\_num\_procs(), earnings);

}

#pragma omp section

{

int earnings = 0;

for (i = 70; i < 100; i++) {

earnings += ORIGINAL\_TICKET\_COST \* 0.9; // 10% off

}

#pragma omp critical

total\_earnings += earnings;

printf("Section 4: Master thread: %d, Processor: %d, Earnings: %d\n", omp\_get\_thread\_num(), omp\_get\_num\_procs(), earnings);

}

}

end\_time = omp\_get\_wtime();

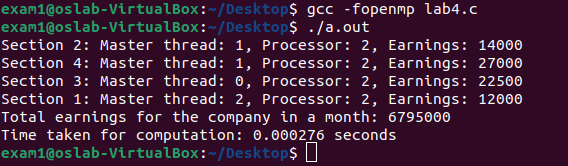
total\_earnings \*= SERVICES\_PER\_DAY \* DAYS\_IN\_MONTH; // Calculate for a month

printf("Total earnings for the company in a month: %d\n", total\_earnings);

printf("Time taken for computation: %f seconds\n", end\_time - start\_time);

return 0;

}



## **CAO Lab Assignment 5**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Title:Private shared variables**

**Problem statement:**

Design a math application:

It accepts an integer number as input and outputs whether it is a rational number, perfect number, or prime number.

Design a parallel program for the same.

- Use sections for every operation.

- Incorporate `private` for rational number, `LastPrivate` for perfect number, and `FirstPrivate` for prime number.

- Limit the number of threads to 3.

#include <stdio.h>

#include <math.h>

#include <omp.h>

int isRational(int n) {

return (n > 0);

}

int isPerfect(int n) {

int sum = 0;

for(int i = 1; i < n; i++){

if(n% i == 0)

sum = sum + i;

}

if(sum == n)

return 1;

else

return 0;

}

int isPrime(int n) {

int i, flag = 0;

if (n <=1)

flag = 1;

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

if (n % i == 0) {

flag = 1;

break;

}

}

if (flag == 0)

return 1;

else

return 0;

}

int main() {

int number;

printf("Enter an integer number: ");

scanf("%d", &number);

int rational, perfect, prime;

omp\_set\_num\_threads(3);

#pragma omp parallel sections private(rational) lastprivate(perfect) firstprivate(prime)

{

#pragma omp section

{

rational = isRational(number);

if (rational)

printf("The number %d is rational.\n", number);

else

printf("The number %d is not a rational.\n", number);

}

#pragma omp section

{

perfect = isPerfect(number);

if (perfect)

printf("The number %d is a perfect number.\n", number);

else

printf("The number %d is not a perfect number.\n", number);

}

#pragma omp section

{

prime = isPrime(number);

if (prime)

printf("The number %d is a prime number.\n", number);

else

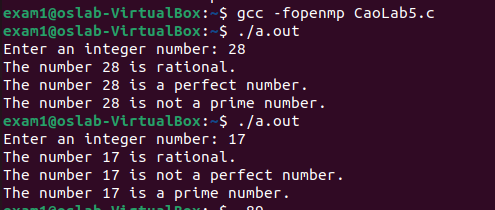
printf("The number %d is not a prime number.\n", number);

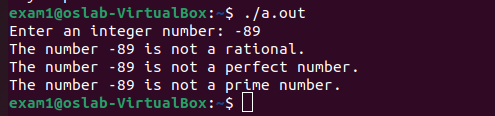
}

}

return 0;

}





## **CAO Lab Assignment 6**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Problem statement:**

The quality checking unit in the toy modeling unit has an incremental counter and counts the tested toy from 0 to 25 . Once the counter reaches the max value all tested toys will be transferred to the dispatching unit in which this counter decrements from the maximum of 25 and reaches to zero. Use last private to get max value. Write an OpenMp program to perform the above scenario using all 3 scheduling concepts.

#include <stdio.h>

#include <omp.h>

#define MAX\_COUNT 25

void quality\_checking\_unit() {

int counter = 0;

#pragma omp parallel for schedule(static, 1) lastprivate(counter)

for (int i = 0; i <= MAX\_COUNT / 3; i++) {

counter = i;

printf("Quality Checking Unit (Static) - Thread %d: Counter = %d\n", omp\_get\_thread\_num(), counter);

}

#pragma omp parallel for schedule(dynamic, 1) lastprivate(counter)

for (int i = MAX\_COUNT / 3 + 1; i <= 2 \* MAX\_COUNT / 3; i++) {

counter = i;

printf("Quality Checking Unit (Dynamic) - Thread %d: Counter = %d\n", omp\_get\_thread\_num(), counter);

}

#pragma omp parallel for schedule(guided, 1) lastprivate(counter)

for (int i = 2 \* MAX\_COUNT / 3 + 1; i <= MAX\_COUNT; i++) {

counter = i;

printf("Quality Checking Unit (Guided) - Thread %d: Counter = %d\n", omp\_get\_thread\_num(), counter);

}

printf("Max value reached in Quality Checking Unit: %d\n", counter);

}

void dispatching\_unit() {

int counter = MAX\_COUNT;

#pragma omp parallel for schedule(static, 1) lastprivate(counter)

for (int i = MAX\_COUNT; i >= 2 \* MAX\_COUNT / 3 + 1; i--) {

counter = i;

printf("Dispatching Unit (Static) - Thread %d: Counter = %d\n", omp\_get\_thread\_num(), counter);

}

#pragma omp parallel for schedule(dynamic, 1) lastprivate(counter)

for (int i = 2 \* MAX\_COUNT / 3; i >= MAX\_COUNT / 3 + 1; i--) {

counter = i;

printf("Dispatching Unit (Dynamic) - Thread %d: Counter = %d\n", omp\_get\_thread\_num(), counter);

}

#pragma omp parallel for schedule(guided, 1) lastprivate(counter)

for (int i = MAX\_COUNT / 3; i >= 0; i--) {

counter = i;

printf("Dispatching Unit (Guided) - Thread %d: Counter = %d\n", omp\_get\_thread\_num(), counter);

}

printf("Min value reached in Dispatching Unit: %d\n", counter);

}

int main() {

printf("Starting Quality Checking Unit...\n");

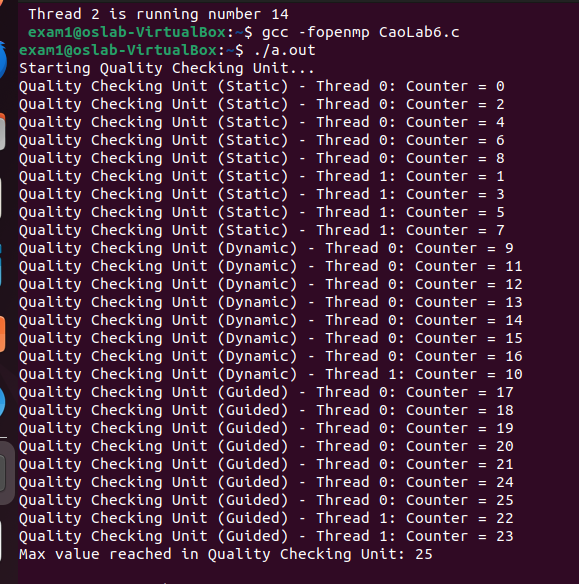
quality\_checking\_unit();

printf("\nStarting Dispatching Unit...\n");

dispatching\_unit();

return 0;

}



## 

## 

## 

## **CAO Lab Assignment 7**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

A] Design a parallel program to print ‘n’ even and odd numbers in sequential fashion of threads.use ordered.

#include <stdio.h>

#include <omp.h>

int main() {

// Set the number of threads

omp\_set\_num\_threads(3);

int value;

printf("Enter a number: ");

scanf("%d", &value);

#pragma omp parallel for ordered

for (int j = 0; j <=value; j++) {

#pragma omp ordered

if (j % 2 == 0) {

printf("thread %d: %d is even \n", omp\_get\_thread\_num(), j);

} else {

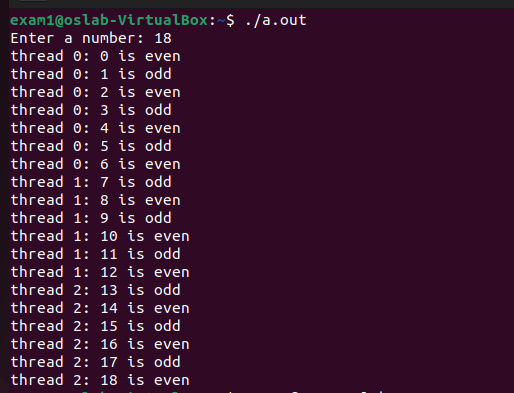
printf("thread %d: %d is odd \n", omp\_get\_thread\_num(), j);

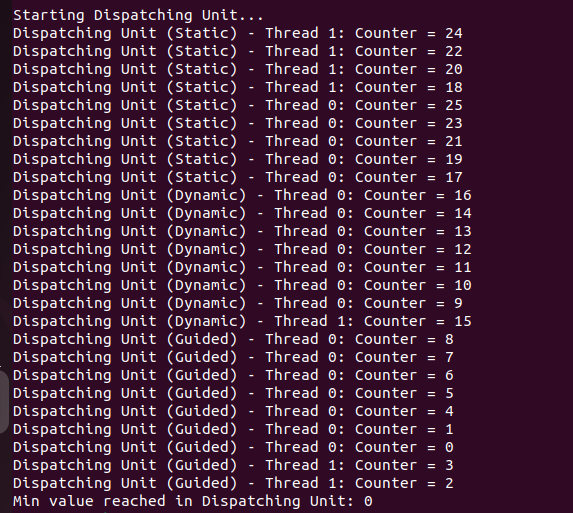
}

}

return 0;

}





B] A contest is being held for TechnoVIT. Students can register, if they want, they can unregister. Registered students (registration numbers:9,3,2…) are stored in an array. Only one student can register or unregister at a time. But they can view the registered list without any constraint. Design a parallel program with the help of locks. Use ordered in any part of the code.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define MAX\_STUDENTS 100

// Global variables

int registered[MAX\_STUDENTS]; // Array to store registered students

int num\_registered = 0; // Number of registered students

omp\_lock\_t lock; // Lock for synchronization

// Function to register a student

void registerStudent(int student\_id) {

omp\_set\_lock(&lock); // Acquire the lock to perform registration

registered[num\_registered] = student\_id;

num\_registered++;

printf("Student %d registered.\n", student\_id);

omp\_unset\_lock(&lock); // Release the lock

}

// Function to unregister a student

void unregisterStudent(int student\_id) {

omp\_set\_lock(&lock); // Acquire the lock to perform unregistration

int found = 0;

for (int i = 0; i < num\_registered; i++) {

if (registered[i] == student\_id) {

// Remove the student from the list by shifting elements

for (int j = i; j < num\_registered - 1; j++) {

registered[j] = registered[j + 1];

}

num\_registered--;

found = 1;

printf("Student %d unregistered.\n", student\_id);

break;

}

}

if (!found) {

printf("Student %d not found in the registered list.\n", student\_id);

}

omp\_unset\_lock(&lock); // Release the lock

}

// Function to display the registered list

void displayRegisteredList() {

omp\_set\_lock(&lock); // Acquire the lock to display the list

printf("Registered students: ");

for (int i = 0; i < num\_registered; i++) {

printf("%d ", registered[i]);

}

printf("\n");

omp\_unset\_lock(&lock); // Release the lock

}

int main() {

int choice, student\_id;

omp\_init\_lock(&lock); // Initialize the lock

while (1) {

printf("Enter your choice:\n");

printf("1. Register\n");

printf("2. Unregister\n");

printf("3. Display registered list\n");

printf("4. Exit\n");

scanf("%d", &choice);

if (choice == 1) {

printf("Enter student ID to register: ");

scanf("%d", &student\_id);

registerStudent(student\_id);

} else if (choice == 2) {

printf("Enter student ID to unregister: ");

scanf("%d", &student\_id);

unregisterStudent(student\_id);

} else if (choice == 3) {

displayRegisteredList();

} else if (choice == 4) {

break;

} else {

printf("Invalid choice. Try again.\n");

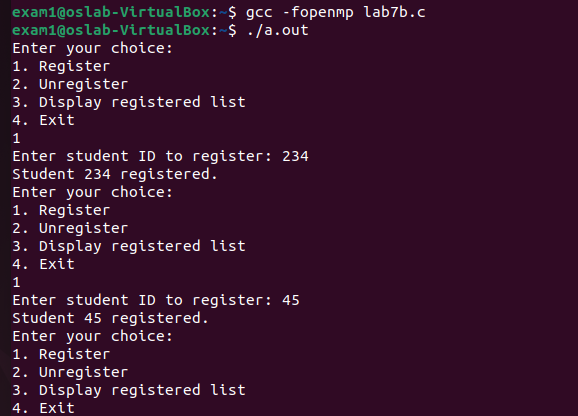
}

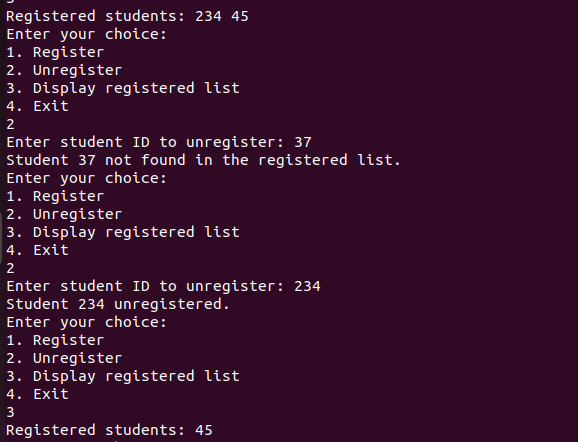
}

omp\_destroy\_lock(&lock); // Destroy the lock

return 0;

}

****

****

## **CAO Lab Assignment 8**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Title:** Barrier\_Series

**Problem statement:**

Write a parallel program using OpenMP to implement the following series,  
1/2 + 1/4 + 1/8 + ...  
Find the sum of the series and print it along with the thread id and the last value in the series for the given “N” value.

Incorporate barrier, scheduling, ordered constructs of OpenMP.

Print the output in a file “series.txt”

Code:

#include <stdio.h>

#include <omp.h>

int main() {

int N;

printf("Reg. no.:24mcs1017\nName:Mahesh Jagtap\n");

printf("Enter the no.of terms for the series 1/2 + 1/4 + 1/8 + ......\n");

scanf("%d",&N);

double sum = 0.0;

#pragma omp parallel num\_threads(4)

{

int tid = omp\_get\_thread\_num();

double thread\_sum = 0.0;

double last\_value = 0.0;

#pragma omp for schedule(static)

for (int i = 1; i <= N; i++) {

double term = 1.0 / (double)(1 << i);

thread\_sum += term;

last\_value = term;

}

// Synchronize all threads before printing

#pragma omp barrier

#pragma omp single

{

printf("Thread %d: Last Value = %lf\n", tid, last\_value);

sum += thread\_sum;

printf("Sum of the series: %lf\n", sum);

// Write the output to the "series.txt" file

FILE \*file = fopen("series.txt", "w");

if (file != NULL) {

fprintf(file, "Thread %d: Last Value = %lf\n", tid, last\_value);

fprintf(file, "Sum of the series: %lf\n", sum);

fclose(file);

}

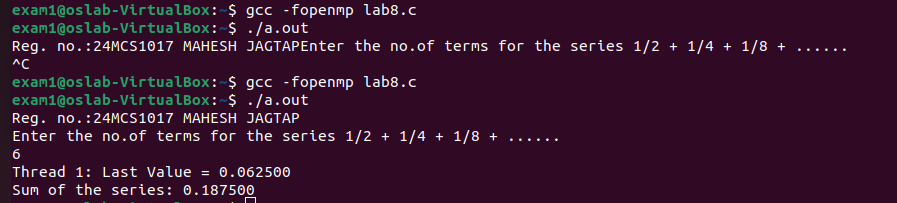
}

}

return 0;

}

Output:

****

## **CAO Lab Assignment 9**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Title:** Matrix Operation

**Problem statement:**

Write a parallel program using OpenMP to Matrix-Vector multiplication

**Code:**

#include <stdio.h>

#include <omp.h>

#include <sys/time.h>

#define N 10 // Define the matrix size (N x N)

int main()

{

struct timeval tv1, tv2;

struct timezone tz;

double elapsed;

int A[N][N], B[N], C[N];

int i, j;

// Initialize matrix A and vector B

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

{

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

{

A[i][j] = i + j;

}

B[i] = i;

C[i] = 0; // Initialize the result vector C

}

gettimeofday(&tv1, &tz);

// Perform Matrix-Vector multiplication using OpenMP parallel for

#pragma omp parallel for shared(A, B, C) private(i, j)

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

{

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

{

C[i] += A[i][j] \* B[j];

}

}

gettimeofday(&tv2, &tz);

elapsed = (double) (tv2.tv\_sec-tv1.tv\_sec) + (double)(tv2.tv\_usec-tv1.tv\_usec) \* 1.e-6;

printf("elapsed time = %f seconds.\n", elapsed);

// Print the result vector C

printf("Result Vector C:\n");

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

{

printf("C[%d] = ",i);

printf("%d", C[i]);

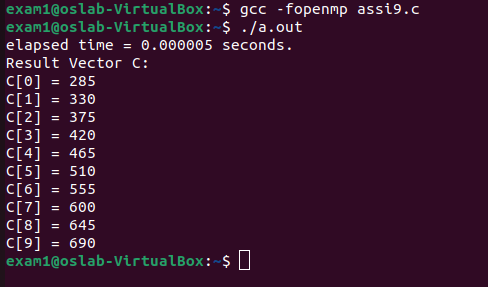
printf("\n");

}

return 0;

}

**Output:**

****

## **CAO Lab Assignment 10**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Problem statement:**

Write a parallel program using OpenMP to Matrix multiplication

**Code:**

#include <stdio.h>

#include<omp.h>

const int N=3;

int main() {

int A[N][N],B[N][N],C[N][N];

// Initialize matrices A and B

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

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

A[i][j] = 1;

B[i][j] = 1;

C[i][j] = 0;

}

}

// Perform matrix multiplication

#pragma omp parallel for

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

#pragma omp parallel for

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

#pragma omp parallel for

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

C[i][j] += A[i][k] \* B[k][j];

}

}

}

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

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

printf("%d\t",C[i][j]);

}

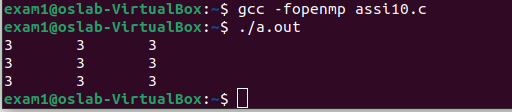
printf("\n");

}

return 0;

}

**Output:**

****

## **CAO Lab Assignment 11**

**Name : Mahesh Jagtap**

**Reg no. : 24MCS1017**

**Title:** Quick\_Sort

**Question:**

Develop a program to analyse the parallel quick sort .

**Code:**

#include <stdio.h>

#include <omp.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return i + 1;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

#pragma omp parallel sections

{

#pragma omp section

quickSort(arr, low, pi - 1);

#pragma omp section

quickSort(arr, pi + 1, high);

}

}

}

int main() {

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr) / sizeof(arr[0]);

#pragma omp parallel

{

#pragma omp single nowait

quickSort(arr, 0, n - 1);

}

printf("Sorted array: ");

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

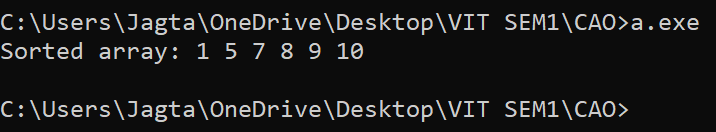
}

printf("\n");

return 0;

}

**Output:**

****