DIGITAL ASSIGNMENT - 3

1. Write an Open MP program using C for the following 1-D array of n-elements and perform the following task. (Find the time for each sub task and print the total time)

1.1. To initialize and print the array elements and to add and multiply a value to each element and print the same.

1.2. To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array.

1.3. To find the sum of square's, cubes of array elements and to find the maximum and minimum in an array and to display the second largest and smallest element in an array.

1.4. To print and count the prime numbers and their sum in the array.

1.5. To find the mean, standard deviation and variance of the given array elements.

1.6. To count the duplicate elements of the array and to find the frequency count of each elements.

Code:

```
#include <stdio.h>
#include <limits.h>
#include <omp.h>
float find_sqrt(float number)
{
   float sqrt=number/2,temp = 0;
   while(sqrt != temp)
   {
      temp = sqrt;
      sqrt = (number/temp + temp) / 2;
   }

   return sqrt;
}
void task1(int array[],int size)
{
   printf("Task1 - To initialize and print the array elements and to add and multiply a value to
each element and print the same");
   int arr1[size],sum=0;
   long int mul=1;
   for(int i=0;i<size;i++)
   {
      arr1[i]=array[i];
      printf("%d\t",arr1[i]);
      sum+=array[i];
      mul*=array[i];
```

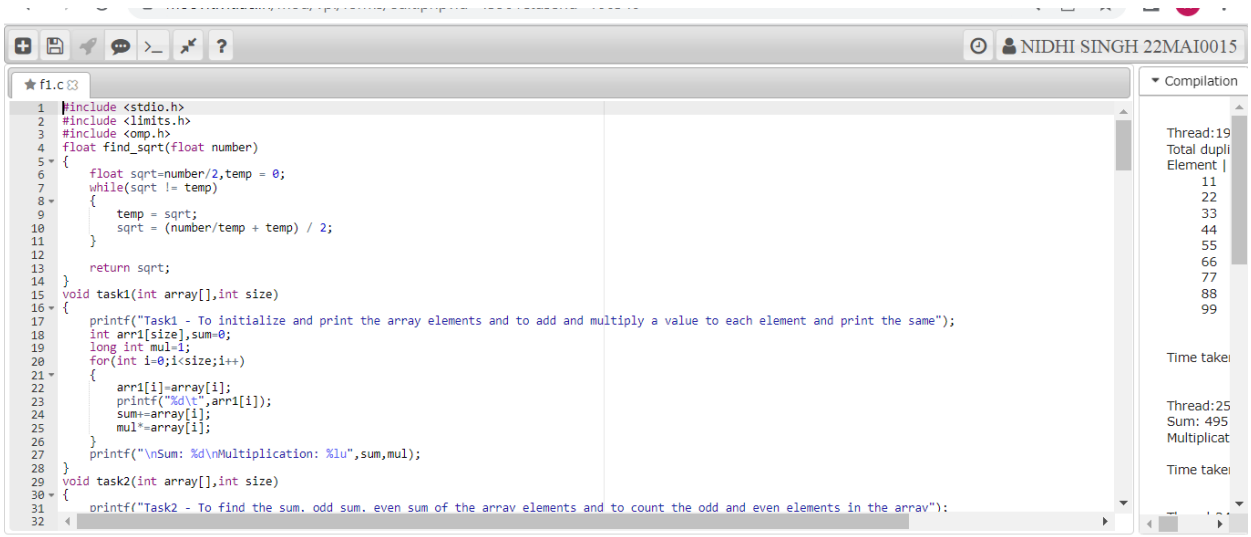
```
    }
    printf("\nSum: %d\nMultiplication: %lu",sum,mul);
}
void task2(int array[],int size)
{
    printf("Task2 - To find the sum, odd sum, even sum of the array elements and to count
the odd and even elements in the array");
    int sum=0,odd_sum=0,even_sum=0,odd_count=0,even_count=0;
    for(int i=0;i<size;i++)
    {
        sum+=array[i];
        if(array[i]%2==0)
        {
            even_sum+=array[i];
            even_count++;
        }
        else
        {
          odd_sum+=array[i];
          odd_count++;
        }

    }
    printf("\nSum: %d\nOdd: %d\nSum of odd: %d\nEven: %d\nSum of even:
%d",sum,odd_count,odd_sum,even_count,even_sum);
}
void task3(int array[],int size)
{
    printf("Task3 - To find the sum of square's, cubes of array elements and to find the
maximum and minimum in an array and to display the second largest and smallest element
in an array");
    int
sum_square=0,sum_cube=0,max=INT_MAX,min=INT_MAX,second_smallest=INT_MAX,seco
nd_largest=INT_MAX;
    for(int i=0;i<size;i++)
    {
        sum_square+=(array[i]*array[i]);
        sum_cube+=(array[i]*array[i]*array[i]);
        if(array[i]<min)
        {
            second_smallest=min;
            min=array[i];
        }
        else if (array[i] < second_smallest && array[i] != min)
        {
            second_smallest = array[i];
```

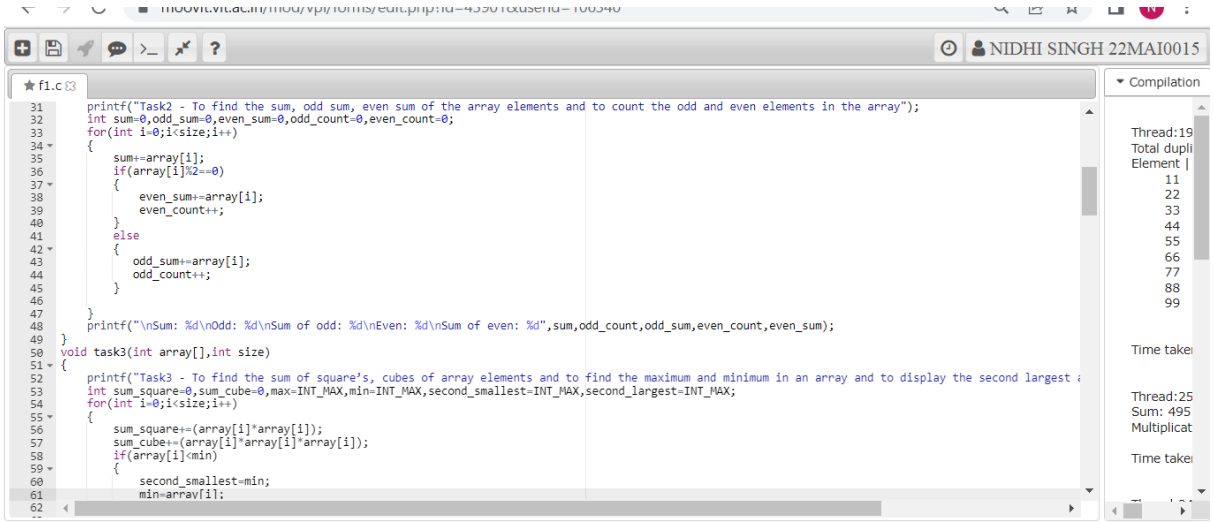
```
      }
      if(array[i]>max)
      {
         second_largest=max;
         max=array[i];
      }
      else if (array[i] > second_largest && array[i] != max)
      {
         second_largest = array[i];
      }
   }

   printf("\nSum of square: %d\nSum of cube: %d\nMaximum: %d\nMinimum: %d\nSecond
largest: %d\nSecond smallest:
%d",sum_square,sum_cube,max,min,second_largest,second_smallest);

}
void task4(int array[],int size)
{
   printf("Task4 - -To print and count the prime numbers and their sum in the array");
   int sum=0,flag=0,count=0;
   for(int i=0;i<size;i++)
   {
      flag=0;
      for(int j=2;j<array[i];j++)
      {
         if(array[i]%j==0)
         {
            flag=1;
            break;
         }
      }
      if(flag==0)
      {
      printf("\t%d",array[i]);
      count++;
      sum+=array[i];
      }
   }
   printf("\nPrime number: %d\nSum of prime no: %d",count,sum);

}
void task5(int array[],int size)
{
   printf("Task5 - To find the mean, standard deviation and variance of the given array
elements");
```

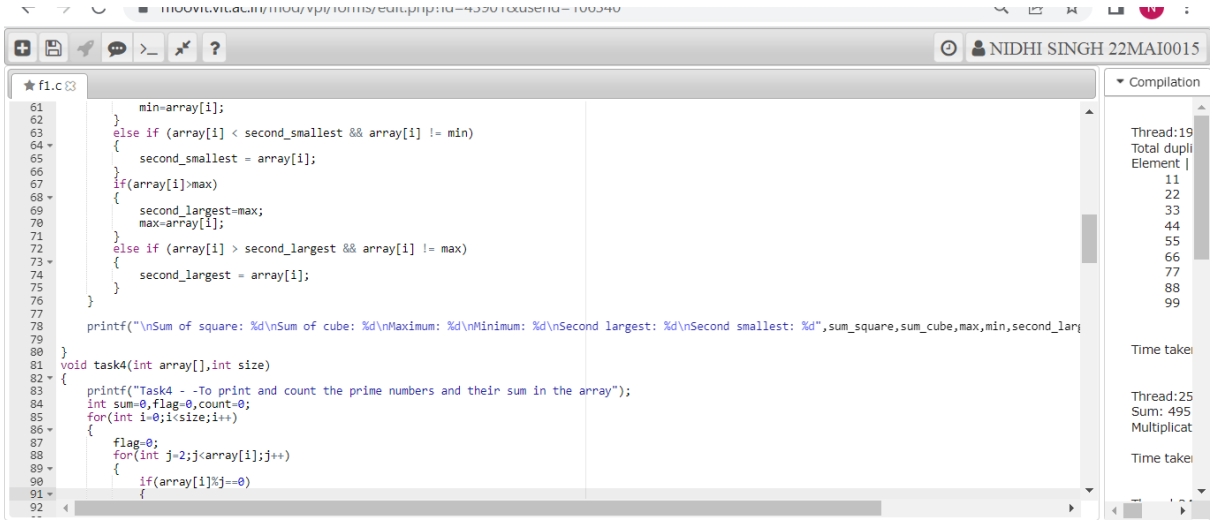
```
    int sum=0,sum1=0;
    float mean,variance,standard_deviation;
    for(int i=0;i<size;i++)
    {
        sum+=array[i];
    }
    mean=sum/(float)size;
    for(int i=0;i<size;i++)
    {
        sum1+=((array[i] - mean)*(array[i] - mean));
    }
    variance = sum1 / (float)size;
    standard_deviation = find_sqrt(variance);
    printf("\nMean: %.2f\nVariance: %.2f\nStandard deviation:
%.2f",mean,variance,standard_deviation);

}
void task6(int array[],int size)
{
    printf("Task6 - To count the duplicate elements of the array and to find the frequency
count of each elements");
    int fr[size],visited = -1,count=0,duplicate=0;
    for(int i = 0; i < size; i++){
        int count = 1;
        for(int j = i+1; j < size; j++){
            if(array[i] == array[j]){
                count++;
                duplicate++;
                fr[j] = visited;
            }
        }
        if(fr[i] != visited)
        {
            fr[i] = count;
        }
    }
    printf("\nTotal duplicate number: %d",duplicate);
    printf("\nElement | Frequency\n");
    for(int i = 0; i < size; i++){
        if(fr[i] != visited){
            printf("\t%d\t|\t%d\n", array[i],fr[i]);
        }
    }

}
```

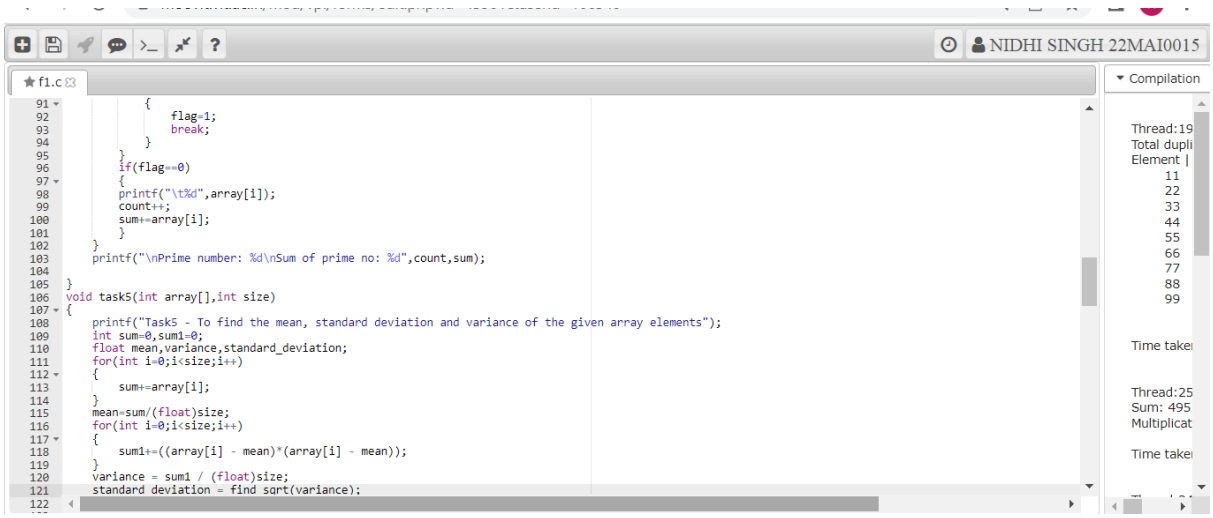
```
int main()
{
   double itime,ftime,total_time;
   int array1[]={11,22,33,44,55,66,77,88,99};
   int size=sizeof(array1)/sizeof(array1[0]);
   itime = omp_get_wtime();

   #pragma omp parallel
   {
      #pragma omp sections
      {
         #pragma omp section
         {
            double start = omp_get_wtime();
            printf("\nThread:%d\t||\t", omp_get_thread_num());
            task1(array1,size);
            double stop = omp_get_wtime();
            double exec_time = stop - start;
            printf("\n\nTime taken for task 1: %f\n\n", exec_time);

         }
         #pragma omp section
         {
            double start = omp_get_wtime();
            printf("\nThread:%d\t||\t",omp_get_thread_num());
            task2(array1,size);
            double stop = omp_get_wtime();
            double exec_time = stop - start;
            printf("\n\nTime taken for task 2: %f\n\n", exec_time);

         }
         #pragma omp section
         {
            double start = omp_get_wtime();
            printf("\nThread:%d\t||\t", omp_get_thread_num());
            task3(array1,size);
            double stop = omp_get_wtime();
            double exec_time = stop - start;
            printf("\n\nTime taken for task 3: %f\n\n", exec_time);

         }
         #pragma omp section
         {
            double start = omp_get_wtime();
            printf("\nThread:%d\t||\t",omp_get_thread_num());
            task4(array1,size);
```

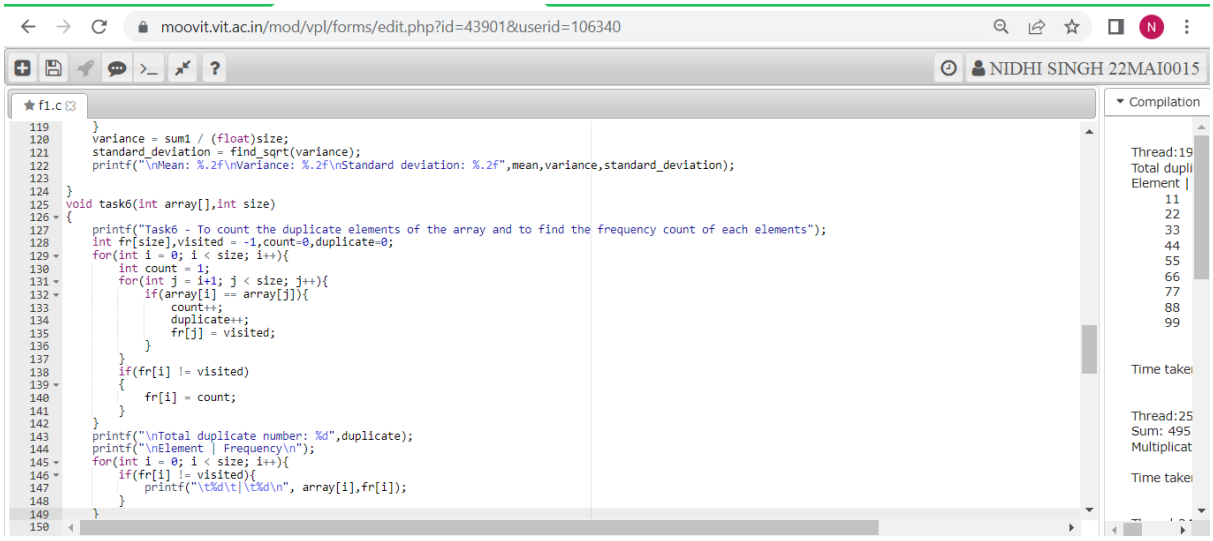
```
            double stop = omp_get_wtime();
            double exec_time = stop - start;
            printf("\n\nTime taken for task 4: %f\n\n", exec_time);

        }
        #pragma omp section
        {
            double start = omp_get_wtime();
            printf("\nThread:%d\t||\t", omp_get_thread_num());
            task5(array1,size);
            double stop = omp_get_wtime();
            double exec_time = stop - start;
            printf("\n\nTime taken for task 5: %f\n\n", exec_time);

        }
        #pragma omp section
        {
            double start = omp_get_wtime();
            printf("\nThread:%d\t||\t",omp_get_thread_num());
            task6(array1,size);
            double stop = omp_get_wtime();
            double exec_time = stop - start;
            printf("\n\nTime taken for task 6: %f\n\n", exec_time);

        }
    }
  }
  ftime = omp_get_wtime();
  total_time = ftime - itime;
  printf("\n\nTime taken for perform the all task: %f\n\n", total_time);
}
```

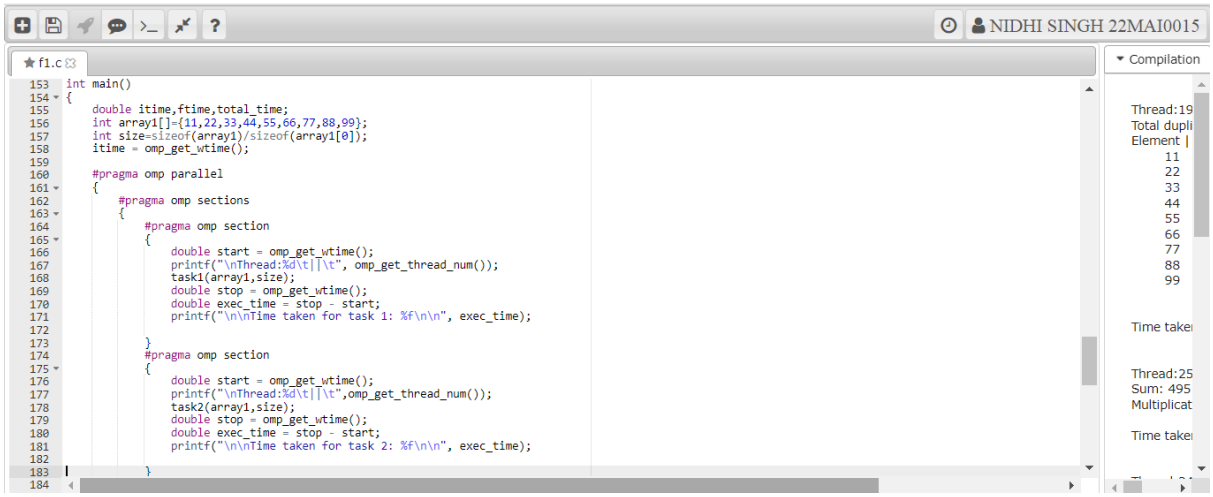


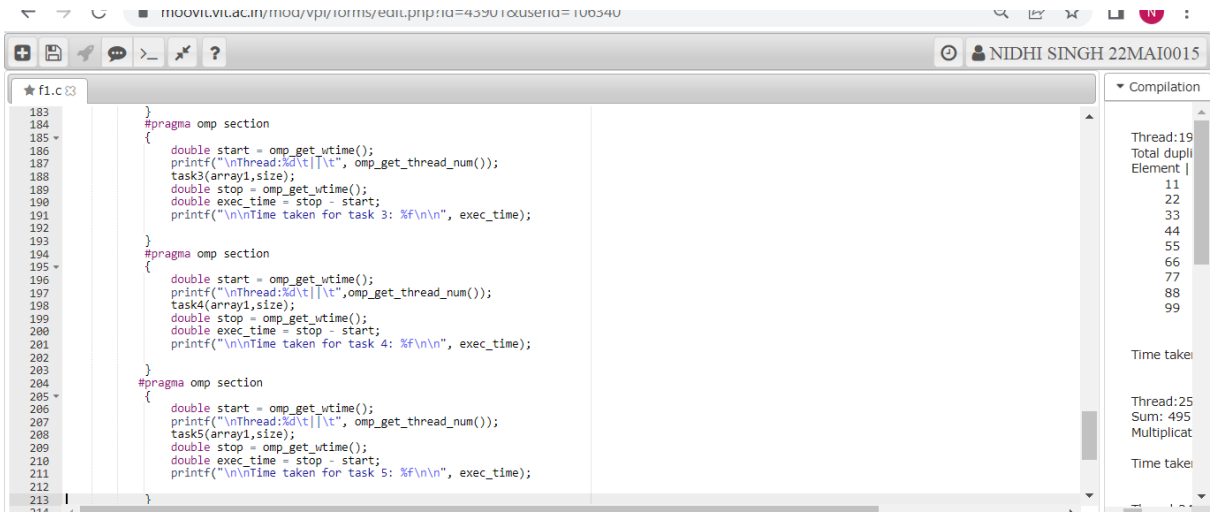

NIDHI SINGH 22MAI0015
f1.c
Compilation
printf("Task2 - To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array");
int sum=0,odd_sum=0,even_sum=0,odd_count=0,even_count=0;
for(int i=0;i<size;i++)
{
sum+=array[i];
if(array[i]%2==0)
{
even_sum+=array[i];
even_count++;
}
else
{
odd_sum+=array[i];
odd_count++;
}
}
printf("\nSum: %d\nOdd: %d\nSum of odd: %d\nEven: %d\nSum of even: %d",sum,odd_count,odd_sum,even_count,even_sum);
}
void task3(int array[],int size)
{
printf("Task3 - To find the sum of square's, cubes of array elements and to find the maximum and minimum in an array and to display the second largest
int sum_square=0,sum_cube=0,max=INT_MAX,min=INT_MAX,second_smallest=INT_MAX,second_largest=INT_MAX;
for(int i=0;i<size;i++)
{
sum_square+=(array[i]*array[i]);
sum_cube+=(array[i]*array[i]*array[i]);
if(array[i]<min)
{
second_smallest=min;
min=array[i];
Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99
Time take
Thread:25
Sum: 495
Multiplicat
Time take

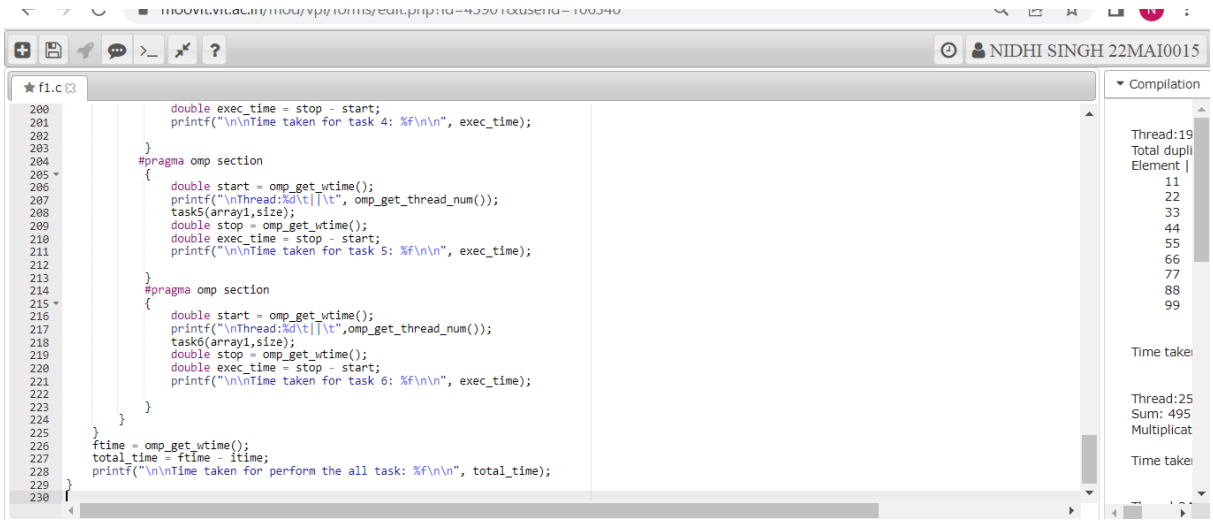

NIDHI SINGH 22MAI0015
f1.c
Compilation
min=array[i];
}
else if (array[i] < second_smallest && array[i] != min)
{
second_smallest = array[i];
}
if(array[i]>max)
{
second_largest=max;
max=array[i];
}
else if (array[i] > second_largest && array[i] != max)
{
second_largest = array[i];
}
}
printf("\nSum of square: %d\nSum of cube: %d\nMaximum: %d\nMinimum: %d\nSecond largest: %d\nSecond smallest: %d",sum_square,sum_cube,max,min,second_larg
}
void task4(int array[],int size)
{
printf("Task4 - -To print and count the prime numbers and their sum in the array");
int sum=0,flag=0,count=0;
for(int i=0;i<size;i++)
{
flag=0;
for(int j=2;j<array[i];j++)
{
if(array[i]%j==0)
{
Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99
Time take
Thread:25
Sum: 495
Multiplicat
Time take


NIDHI SINGH 22MAI0015
f1.c
Compilation
{
flag=1;
break;
}
}
if(flag==0)
{
printf("\t%d",array[i]);
count++;
sum+=array[i];
}
}
printf("\nPrime number: %d\nSum of prime no: %d",count,sum);
}
void task5(int array[],int size)
{
printf("Task5 - To find the mean, standard deviation and variance of the given array elements");
int sum=0,sum1=0;
float mean,variance,standard_deviation;
for(int i=0;i<size;i++)
{
sum+=array[i];
}
mean=sum/(float)size;
for(int i=0;i<size;i++)
{
sum1+=((array[i] - mean)*(array[i] - mean));
}
variance = sum1 / (float)size;
standard_deviation = find sqrt(variance);
Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99
Time take
Thread:25
Sum: 495
Multiplicat
Time take


moovit.vit.ac.in/mod/vpl/forms/edit.php?id=43901&userid=106340
NIDHI SINGH 22MAI0015
f1.c
Compilation
}
variance = sum1 / (float)size;
standard_deviation = find_sqrt(variance);
printf("\nMean: %.2f\nVariance: %.2f\nStandard deviation: %.2f",mean,variance,standard_deviation);
}
void task6(int array[],int size)
{
printf("Task6 - To count the duplicate elements of the array and to find the frequency count of each elements");
int fr[size],visited = -1,count=0,duplicate=0;
for(int i = 0; i < size; i++){
int count = 1;
for(int j = i+1; j < size; j++){
if(array[i] == array[j]){
count++;
duplicate++;
fr[j] = visited;
}
}
if(fr[i] != visited)
{
fr[i] = count;
}
}
printf("\nTotal duplicate number: %d",duplicate);
printf("\nElement | Frequency\n");
for(int i = 0; i < size; i++){
if(fr[i] != visited){
printf("\t%d\t|\t%d\n", array[i],fr[i]);
}
}
}
Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99
Time take
Thread:25
Sum: 495
Multiplicat
Time take

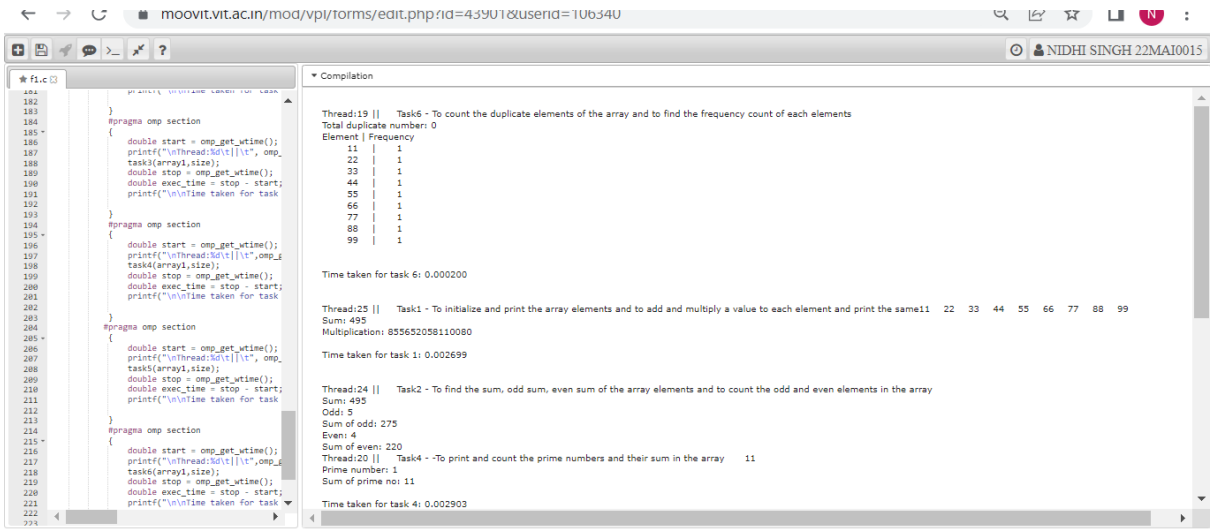

NIDHI SINGH 22MAI0015
f1.c
Compilation
int main()
{
double itime,ftime,total_time;
int array1[]={11,22,33,44,55,66,77,88,99};
int size=sizeof(array1)/sizeof(array1[0]);
itime = omp_get_wtime();
#pragma omp parallel
{
#pragma omp sections
{
#pragma omp section
{
double start = omp_get_wtime();
printf("\nThread:%d\t|\t", omp_get_thread_num());
task1(array1,size);
double stop = omp_get_wtime();
double exec_time = stop - start;
printf("\n\nTime taken for task 1: %f\n\n", exec_time);
}
#pragma omp section
{
double start = omp_get_wtime();
printf("\nThread:%d\t|\t",omp_get_thread_num());
task2(array1,size);
double stop = omp_get_wtime();
double exec_time = stop - start;
printf("\n\nTime taken for task 2: %f\n\n", exec_time);
}
Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99
Time take
Thread:25
Sum: 495
Multiplicat
Time take


NIDHI SINGH 22MAI0015
f1.c
Compilation
}
#pragma omp section
{
double start = omp_get_wtime();
printf("\nThread:%d\t|\t", omp_get_thread_num());
task3(array1,size);
double stop = omp_get_wtime();
double exec_time = stop - start;
printf("\n\nTime taken for task 3: %f\n\n", exec_time);
}
#pragma omp section
{
double start = omp_get_wtime();
printf("\nThread:%d\t|\t",omp_get_thread_num());
task4(array1,size);
double stop = omp_get_wtime();
double exec_time = stop - start;
printf("\n\nTime taken for task 4: %f\n\n", exec_time);
}
#pragma omp section
{
double start = omp_get_wtime();
printf("\nThread:%d\t|\t", omp_get_thread_num());
task5(array1,size);
double stop = omp_get_wtime();
double exec_time = stop - start;
printf("\n\nTime taken for task 5: %f\n\n", exec_time);
}
Thread:19
Total dupli
Element |
11
22
33
44
55
66
77
88
99
Time take
Thread:25
Sum: 495
Multiplicat
Time take

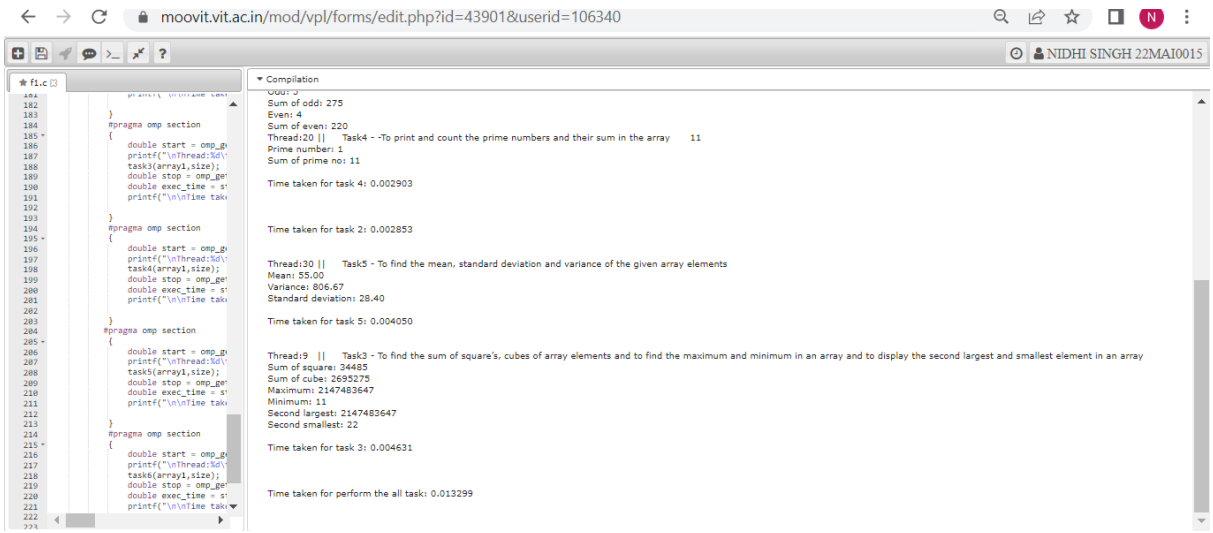


```
                double exec_time = stop - start;
                printf("\n\nTime taken for task 4: %f\n\n", exec_time);

            }
            #pragma omp section
            {
                double start = omp_get_wtime();
                printf("\nThread:%d\t||\t", omp_get_thread_num());
                task5(array1,size);
                double stop = omp_get_wtime();
                double exec_time = stop - start;
                printf("\n\nTime taken for task 5: %f\n\n", exec_time);

            }
            #pragma omp section
            {
                double start = omp_get_wtime();
                printf("\nThread:%d\t||\t",omp_get_thread_num());
                task6(array1,size);
                double stop = omp_get_wtime();
                double exec_time = stop - start;
                printf("\n\nTime taken for task 6: %f\n\n", exec_time);

            }
        }
    }
    ftime = omp_get_wtime();
    total_time = ftime - itime;
    printf("\n\nTime taken for perform the all task: %f\n\n", total_time);
}
```

Output:-



```
Thread:19 ||    Task6 - To count the duplicate elements of the array and to find the frequency count of each elements
Total duplicate number: 0
Element | Frequency
      11    |    1
      22    |    1
      33    |    1
      44    |    1
      55    |    1
      66    |    1
      77    |    1
      88    |    1
      99    |    1

Time taken for task 6: 0.000200

Thread:25 ||    Task1 - To initialize and print the array elements and to add and multiply a value to each element and print the same11   22   33   44   55   66   77   88   99
Sum: 495
Multiplication: 855652058110080

Time taken for task 1: 0.002699

Thread:24 ||    Task2 - To find the sum, odd sum, even sum of the array elements and to count the odd and even elements in the array
Sum: 495
Odd: 5
Sum of odd: 275
Even: 4
Sum of even: 220
Thread:20 ||    Task4 - -To print and count the prime numbers and their sum in the array      11
Prime number: 1
Sum of prime no: 11

Time taken for task 4: 0.002903
```

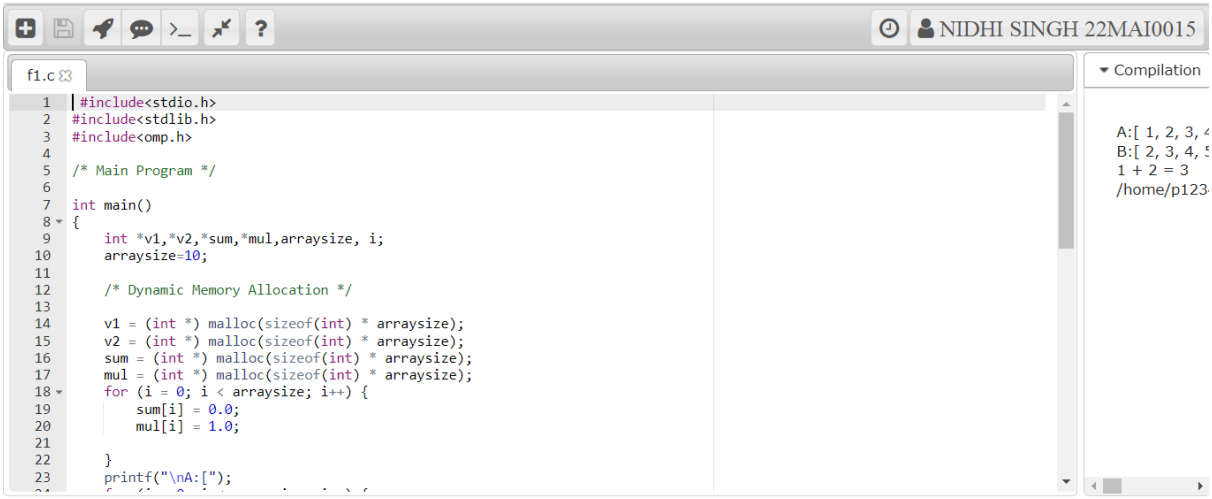


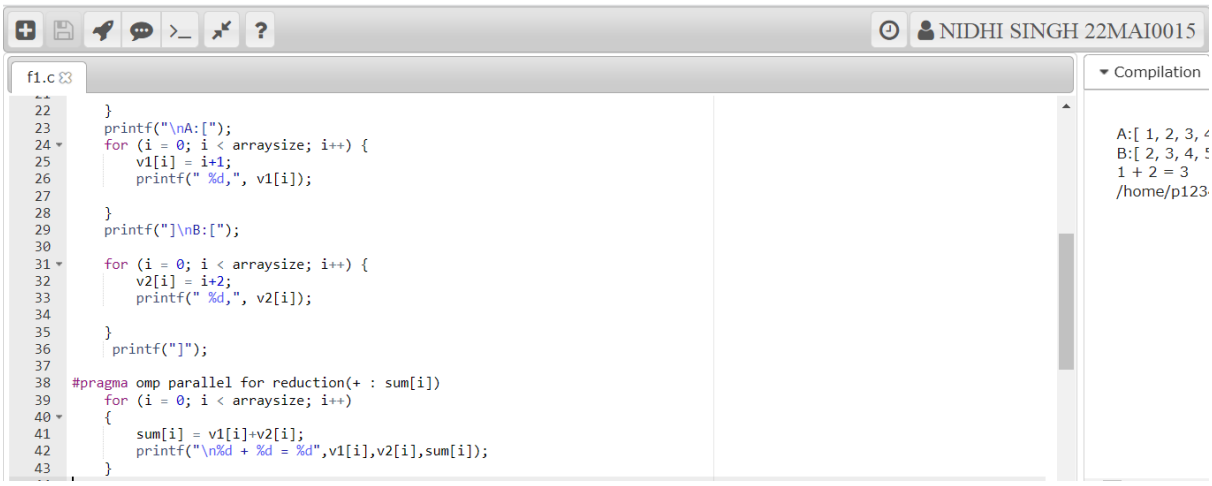
```
Sum of odd: 275
Even: 4
Sum of even: 220
Thread:20 ||    Task4 - -To print and count the prime numbers and their sum in the array      11
Prime number: 1
Sum of prime no: 11

Time taken for task 4: 0.002903


Time taken for task 2: 0.002853


Thread:30 ||    Task5 - To find the mean, standard deviation and variance of the given array elements
Mean: 55.00
Variance: 806.67
Standard deviation: 28.40

Time taken for task 5: 0.004050


Thread:9 ||    Task3 - To find the sum of square's, cubes of array elements and to find the maximum and minimum in an array and to display the second largest and smallest element in an array
Sum of square: 34485
Sum of cube: 2695275
Maximum: 2147483647
Minimum: 11
Second largest: 2147483647
Second smallest: 22

Time taken for task 3: 0.004631


Time taken for perform the all task: 0.013299
```

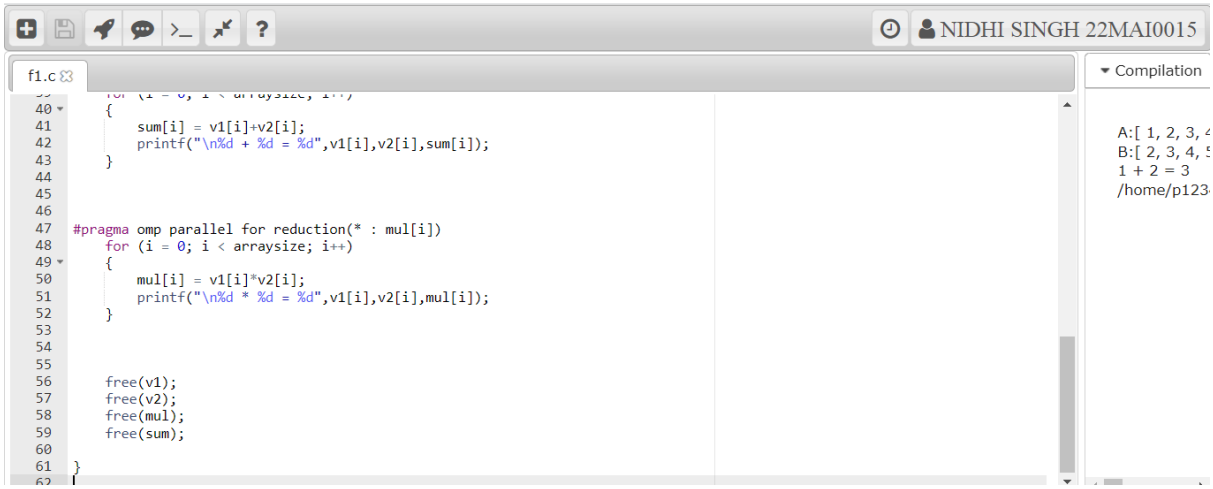
2. **Find the vector sum and vector product of n– elements in an array using open MP reduction clause.**

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

/* Main Program */

int main()
{
        int *v1,*v2,*sum,*mul,arraysize, i;
   arraysize=10;

        /* Dynamic Memory Allocation */

        v1 = (int *) malloc(sizeof(int) * arraysize);
        v2 = (int *) malloc(sizeof(int) * arraysize);
        sum = (int *) malloc(sizeof(int) * arraysize);
        mul = (int *) malloc(sizeof(int) * arraysize);
        for (i = 0; i < arraysize; i++) {
                sum[i] = 0.0;
                mul[i] = 1.0;

        }
   printf("\nA:[");
        for (i = 0; i < arraysize; i++) {
                v1[i] = i+1;
                printf(" %d,", v1[i]);

        }
   printf("]\nB:[");

        for (i = 0; i < arraysize; i++) {
                v2[i] = i+2;
                printf(" %d,", v2[i]);
 }
    printf("]");

#pragma omp parallel for reduction(+ : sum[i])
        for (i = 0; i < arraysize; i++)
        {
                sum[i] = v1[i]+v2[i];
                printf("\n%d + %d = %d",v1[i],v2[i],sum[i]);
        }
#pragma omp parallel for reduction(* : mul[i])
```

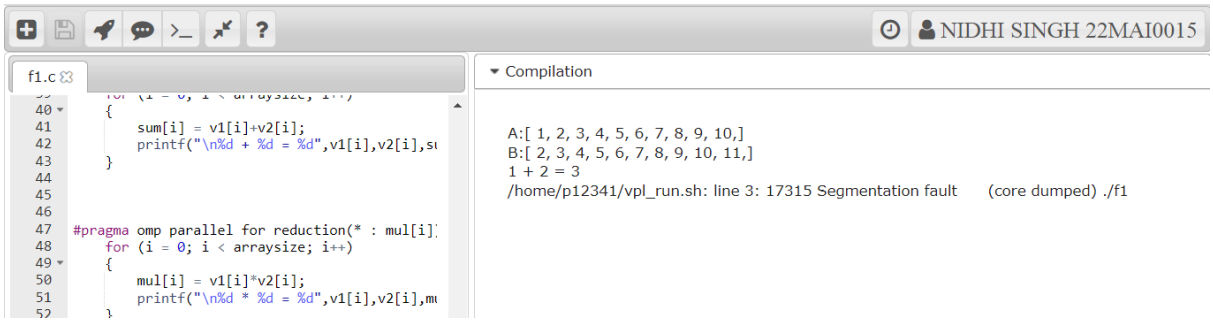
```
        for (i = 0; i < arraysize; i++)
        {
                mul[i] = v1[i]*v2[i];
                printf("\n%d * %d = %d",v1[i],v2[i],mul[i]);
  }
         free(v1);
        free(v2);
        free(mul);
        free(sum);

}
```

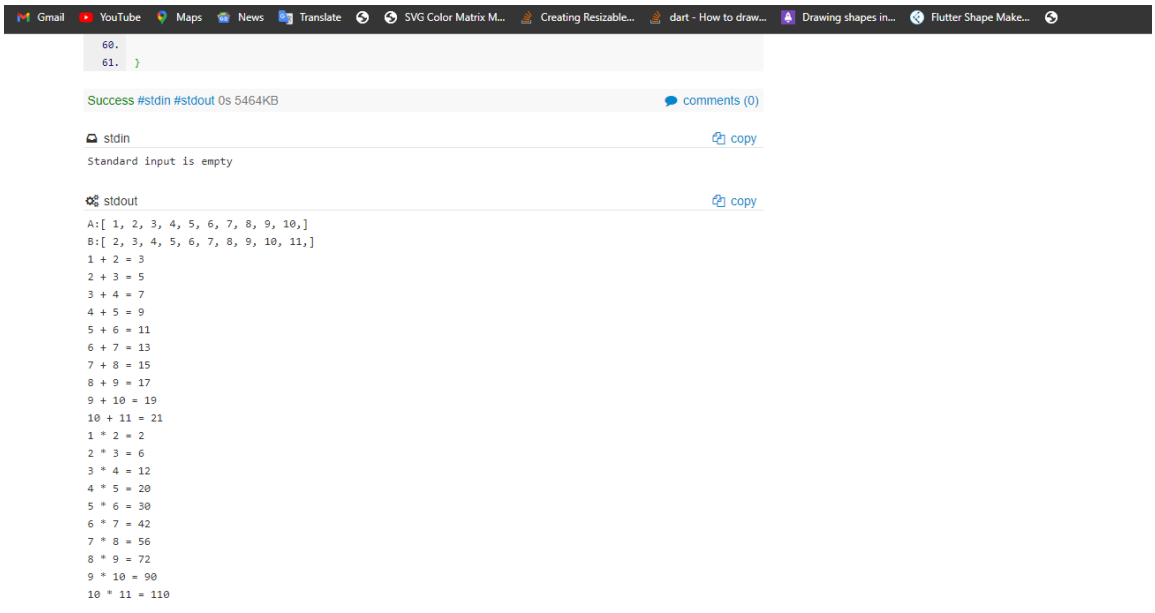





NIDHI SINGH 22MAI0015
f1.c
Compilation
{
sum[i] = v1[i]+v2[i];
printf("\n%d + %d = %d",v1[i],v2[i],sum[i]);
}
#pragma omp parallel for reduction(* : mul[i])
for (i = 0; i < arraysize; i++)
{
mul[i] = v1[i]*v2[i];
printf("\n%d * %d = %d",v1[i],v2[i],mul[i]);
}
free(v1);
free(v2);
free(mul);
free(sum);
}

Output:-


NIDHI SINGH 22MAI0015
f1.c
Compilation
A:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]
B:[2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]
1 + 2 = 3
/home/p12341/vpl_run.sh: line 3: 17315 Segmentation fault (core dumped) ./f1


Gmail YouTube Maps News Translate SVG Color Matrix M... Creating Resizable... dart - How to draw... Drawing shapes in... Flutter Shape Make...
Success #stdin #stdout 0s 5464KB
comments (0)
stdin
copy
Standard input is empty
stdout
copy
A:[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]
B:[2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]
1 + 2 = 3
2 + 3 = 5
3 + 4 = 7
4 + 5 = 9
5 + 6 = 11
6 + 7 = 13
7 + 8 = 15
8 + 9 = 17
9 + 10 = 19
10 + 11 = 21
1 * 2 = 2
2 * 3 = 6
3 * 4 = 12
4 * 5 = 20
5 * 6 = 30
6 * 7 = 42
7 * 8 = 56
8 * 9 = 72
9 * 10 = 90
10 * 11 = 110