NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY

IT 301 Parallel Computing LAB 2

19th August 2020

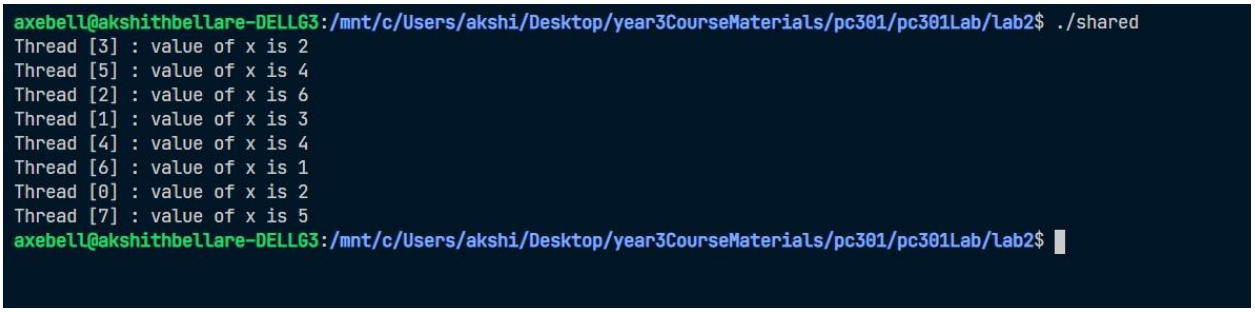
Faculty: Dr. Geetha V and Mrs. Tanmayee

Execute following programs and put screen shots of the output. Write analysis of the result before uploading in IRIS as a single pdf file. for programming exercises, write the code and also put screenshot of the results.

1. Program 1

Aim: To understand and analyze shared clause in parallel directive.

```
/*shared.c*/
#include<omp.h>
int main()
{
int x=0;
#pragma omp parallel shared(x)
{
int tid=omp_get_thread_num();
x=x+1;
printf("Thread [%d]\n value of x is %d",tid,x);
}
}
```



```
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$ ./shared
Thread [3] : value of x is 2
Thread [5] : value of x is 4
Thread [2] : value of x is 6
Thread [1] : value of x is 3
Thread [4] : value of x is 4
Thread [6] : value of x is 1
Thread [0] : value of x is 2
Thread [7] : value of x is 5
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$
```

X is shared among all the threads. Two threads might increment X at the same time.

For example in the above screenshot thead 6 sets X to 1 and then thread 3 and thread 0 both increment X to 2 at the same time.

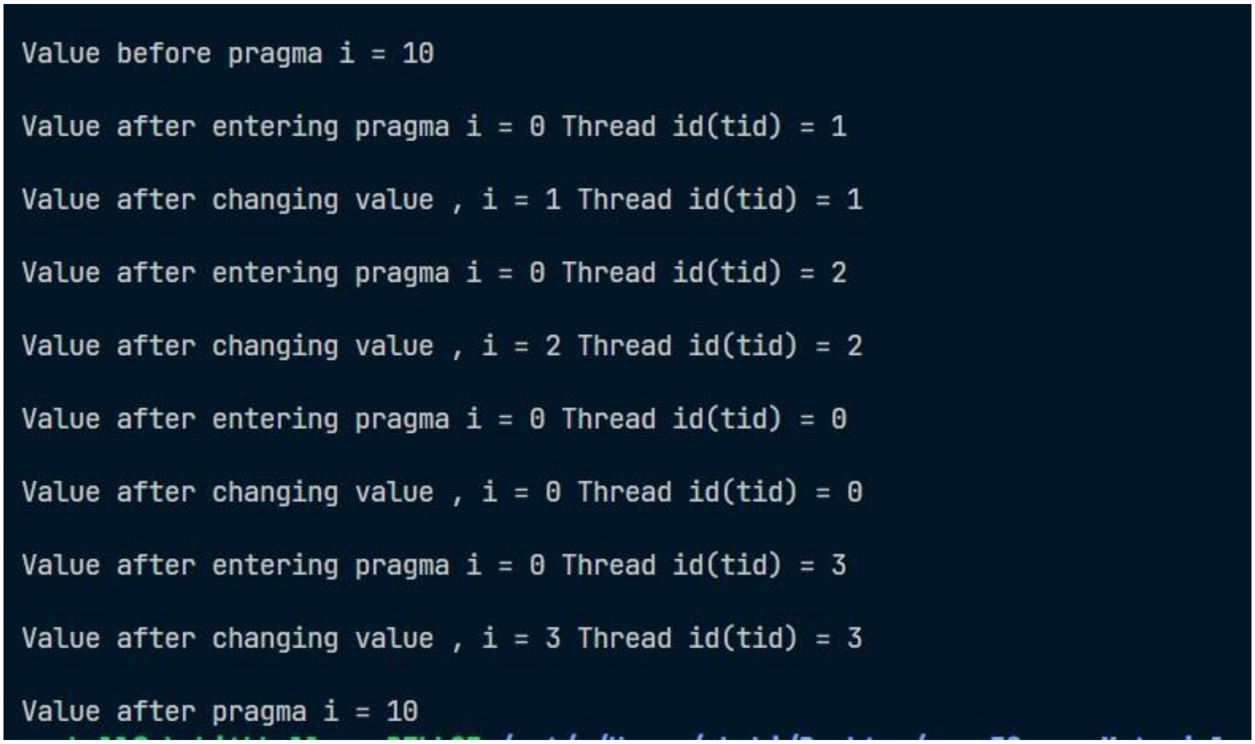
2. Program 2

Learn the concept of private(), firstprivate()

```
/*learn.c*/
```

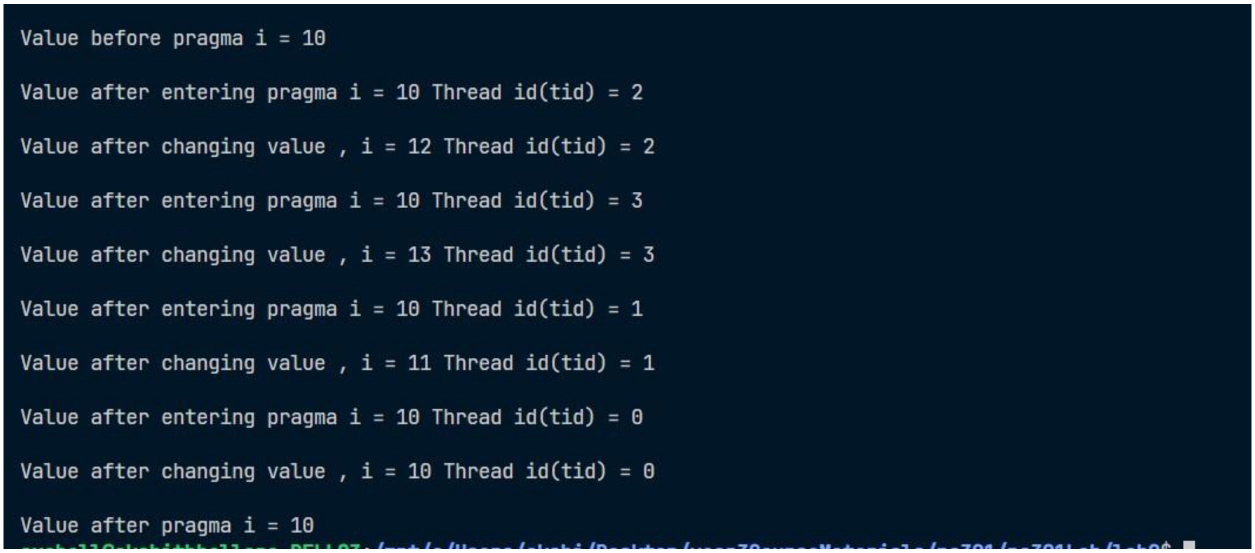
```
#include<stdio.h>
#include<omp.h>
int main()
{
int i=10;
printf("Value before pragma i=%d\n",i);
#pragma omp parallel num_threads(4) private(i)
{

printf("Value after entering pragma i=%d tid=%d\n",i, omp_get_thread_num());
i=i+omp_get_thread_num(); //adds thread_id to i
printf("Value after changing value i=%d tid=%d\n",i, omp_get_thread_num());
}
printf("Value after having pragma i=%d tid=%d\n",i, omp_get_thread_num());
}
```

*** Note down the result by changing private() to firstprivate().**



```
Value before pragma i = 10

Value after entering pragma i = 0 Thread id(tid) = 1

Value after changing value , i = 1 Thread id(tid) = 1

Value after entering pragma i = 0 Thread id(tid) = 2

Value after changing value , i = 2 Thread id(tid) = 2

Value after entering pragma i = 0 Thread id(tid) = 0

Value after changing value , i = 0 Thread id(tid) = 0

Value after entering pragma i = 0 Thread id(tid) = 3

Value after changing value , i = 3 Thread id(tid) = 3

Value after pragma i = 10
```

- Declaring i as private makes i private to each thread so that each thread has a copy of i with value 0 and not 10.Each thread then increments the value of i to i + thread_id.



```
Value before pragma i = 10

Value after entering pragma i = 10 Thread id(tid) = 2

Value after changing value , i = 12 Thread id(tid) = 2

Value after entering pragma i = 10 Thread id(tid) = 3

Value after changing value , i = 13 Thread id(tid) = 3

Value after entering pragma i = 10 Thread id(tid) = 1

Value after changing value , i = 11 Thread id(tid) = 1

Value after entering pragma i = 10 Thread id(tid) = 0

Value after changing value , i = 10 Thread id(tid) = 0

Value after pragma i = 10
```

When i is declared to be firstprivate a copy of the variable i is given to each thread with initial value that it had before the parallel region.

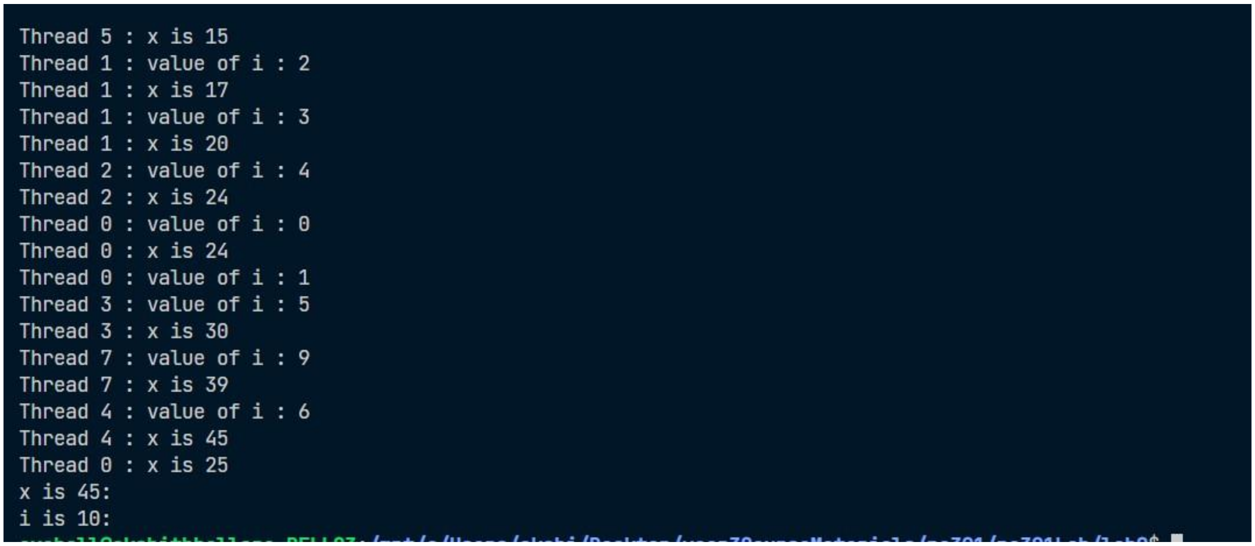
In private case local copy of i had the value 0.

In the firstprivate case it had a value of 10.

3. Program 3

Learn the working of lastprivate() clause:

```
#include<stdio.h>
#include<omp.h>
void main()
{ int x=0,i,n;
printf("Enter the value of n");
scanf("%d",&n);
#pragma omp parallel
{
int id=omp_get_thread_num();
#pragma omp for lastprivate(i)
for(i=0;i<n;i++)
{
printf("Thread %d: value of i : %d\n",id,i);
x=x+i;
printf("Thread %d: x is %d\n",id,x);
}
}
printf("x is %d\n",x);
printf("i IS %d\n",i);
}
```

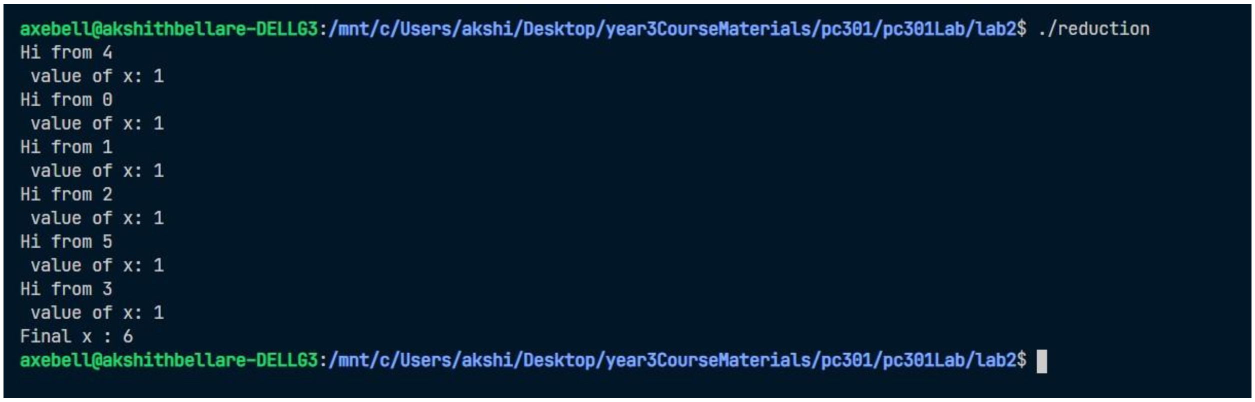


```
Thread 5 : x is 15
Thread 1 : value of i : 2
Thread 1 : x is 17
Thread 1 : value of i : 3
Thread 1 : x is 20
Thread 2 : value of i : 4
Thread 2 : x is 24
Thread 0 : value of i : 0
Thread 0 : x is 24
Thread 0 : value of i : 1
Thread 3 : value of i : 5
Thread 3 : x is 30
Thread 7 : value of i : 9
Thread 7 : x is 39
Thread 4 : value of i : 6
Thread 4 : x is 45
Thread 0 : x is 25
x is 45:
i is 10:
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$
```

* Lastprivate makes the variable private to each thread but the final value of the variable is set to the private version of whichever thread executes the final iteration.

4. Demonstration of reduction clause in parallel directive.

```
#include<stdio.h>
#include<omp.h>
void main()
{
int x=0;
#pragma omp parallel num_threads(6) reduction(+:x)
{
int id=omp_get_thread_num();
int threads=omp_get_num_threads();
x=x+1;
printf("Hi from %d\n value of x : %d\n",id,x);
}
printf("Final x:%d\n",x);
}
```



```
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$ ./reduction
Hi from 4
 value of x: 1
Hi from 0
 value of x: 1
Hi from 1
 value of x: 1
Hi from 2
 value of x: 1
Hi from 5
 value of x: 1
Hi from 3
 value of x: 1
Final x : 6
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$
```

* Reduction clause takes the form of reduction(operator: list of variables that is operated by the operator)

In the example above we are incrementing x. Each thread gets a local copy of x and it performs its operation

Finally all the local copies are combined into a single value using the operator in consideration.

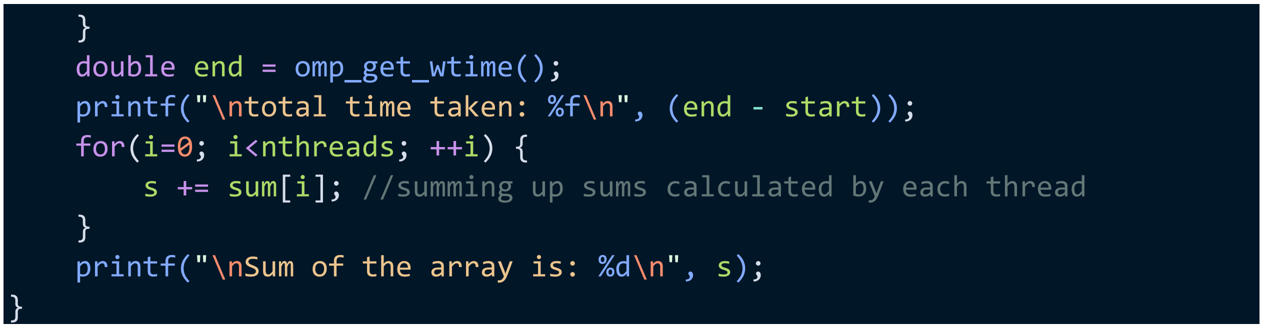
Then this single value is assigned to the global value.

5. Programming exercise

1. Write a parallel program to calculate the sum of elements in an array

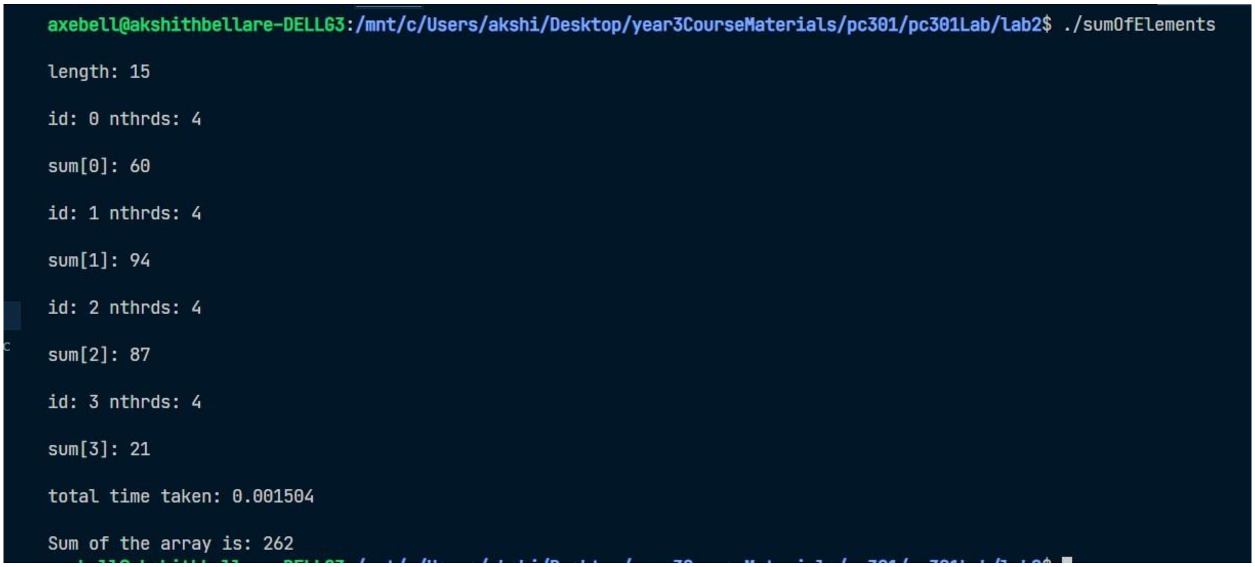
Code

```
#include<omp.h>

#include<stdio.h>

#define NUM_THREADS 4
int main() {
    int a[] = {1,4,1,3,4,1,5,6,1,22,3,12,54,67,78};
    int len = sizeof(a)/sizeof(a[0]);
    printf("\nlength: %d\n", len);
    int sum[NUM_THREADS]; //array to store the sum calculated by each thre
ad.
    int s = 0; //holds the sum of sums calculated by each thread
    int i,nthreads;
    omp_set_num_threads(NUM_THREADS);
    double start = omp_get_wtime();
    #pragma omp parallel
    {
        int i,id ,nthrds;
        id = omp_get_thread_num();
        nthrds = omp_get_num_threads();
        printf("\nid: %d nthrds: %d\n", id, nthrds);
        //if master thread set the number of threads. Threads allocated mi
ght be lesser than what we set.
        if(id ==0) nthreads = nthrds;
        //calculating sum in a round robin approach
        for(i=id, sum[id]=0; i<len; i+=nthrds) {
            sum[id] += a[i];
        }
        printf("\nsum[%d]: %d\n", id, sum[id]);
```



```
    }
    double end = omp_get_wtime();
    printf("\ntotal time taken: %f\n", (end - start));
    for(i=0; i<nthreads; ++i) {
        s += sum[i]; //summing up sums calculated by each thread
    }
    printf("\nSum of the array is: %d\n", s);
}
```

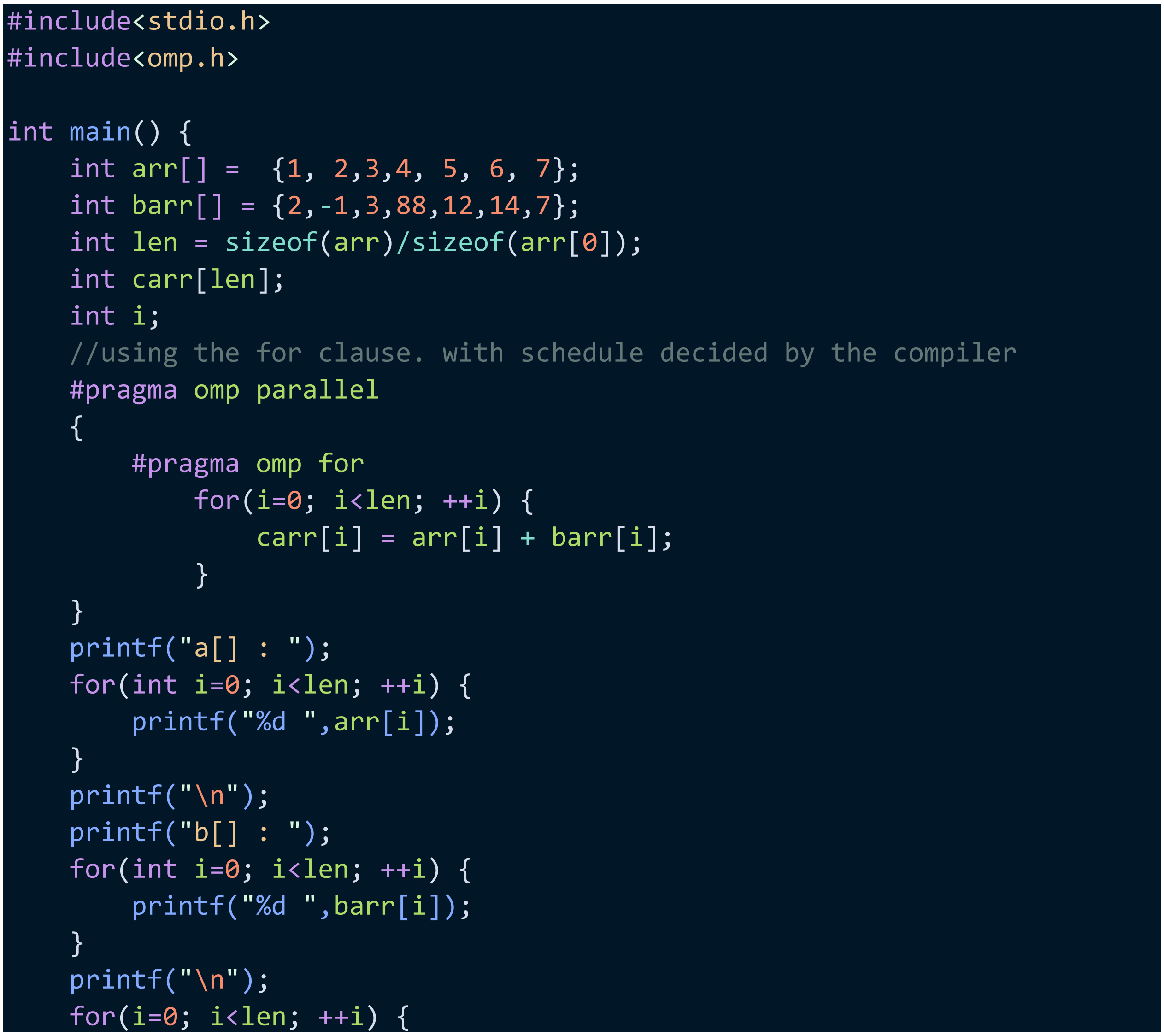
Screenshots



```
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$ ./sumOfElements

length: 15

id: 0 nthrds: 4

sum[0]: 60

id: 1 nthrds: 4

sum[1]: 94

id: 2 nthrds: 4

sum[2]: 87

id: 3 nthrds: 4

sum[3]: 21

total time taken: 0.001504

Sum of the array is: 262
```

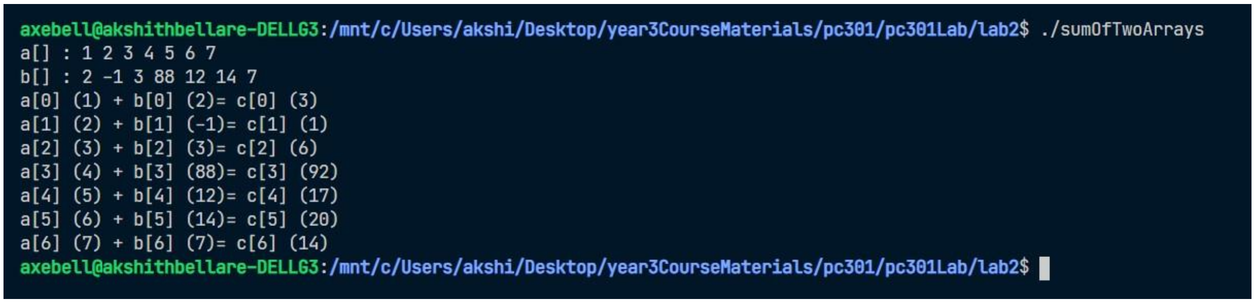
2. Write a parallel program to calculate the a[i]=b[i]+c[i], for all elements in array b[] and c[]

Code



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

int main() {
    int arr[] =  {1, 2,3,4, 5, 6, 7};
    int barr[] = {2,-1,3,88,12,14,7};
    int len = sizeof(arr)/sizeof(arr[0]);
    int carr[len];
    int i;
    //using the for clause. with schedule decided by the compiler
    #pragma omp parallel
    {
        #pragma omp for
            for(i=0; i<len; ++i) {
                carr[i] = arr[i] + barr[i];
            }
    }
    printf("a[] : ");
    for(int i=0; i<len; ++i) {
        printf("%d ",arr[i]);
    }
    printf("\n");
    printf("b[] : ");
    for(int i=0; i<len; ++i) {
        printf("%d ",barr[i]);
    }
    printf("\n");
    for(i=0; i<len; ++i) {
        printf("a[%d] (%d) + b[%d] (%d)= c[%d] (%d)\n ",i, arr[i], i, barr
[i],i , carr[i]);
    }
}
```

Screenshots



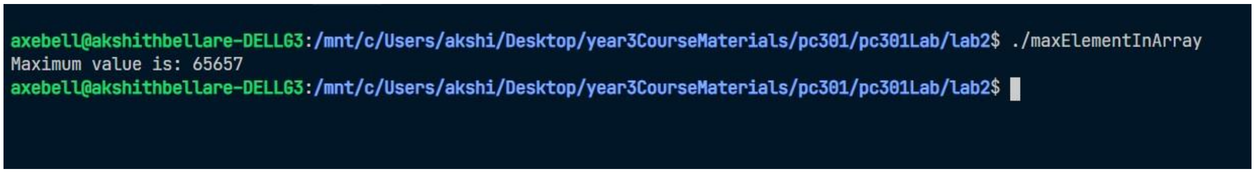
```
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$ ./sumOfTwoArrays
a[] : 1 2 3 4 5 6 7
b[] : 2 -1 3 88 12 14 7
a[0] (1) + b[0] (2)= c[0] (3)
a[1] (2) + b[1] (-1)= c[1] (1)
a[2] (3) + b[2] (3)= c[2] (6)
a[3] (4) + b[3] (88)= c[3] (92)
a[4] (5) + b[4] (12)= c[4] (17)
a[5] (6) + b[5] (14)= c[5] (20)
a[6] (7) + b[6] (7)= c[6] (14)
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$
```

3. Write a parallel program to find the largest among all elements in an array.

Code

```
      #include <stdio.h>

#include <omp.h>
#include <limits.h>
int main() {
    int arr[] = {1,2,4,5,11,2,89,12,66,123,2,1234,34,65657,12,3545,12,334,
12,-12,23};
    int len = sizeof(arr)/sizeof(arr[0]);
    int max_value = INT_MIN; //set to INT_MIN as the maximum value has to
be found
    int i;
    //schedule and chunk size taken from OMP_SCHEDULE env variable
    //reduction used with max operator
    #pragma omp parallel for reduction(max: max_value) schedule(runtime)
        for(i=0; i<len; ++i) {
            if(arr[i] > max_value) {
                max_value = arr[i];
            }
        }
    //reduction clause compares the local copies of each thread and assign
e the maximum value out of those
    //copies to the global max_value
    printf("Maximum value is: %d\n", max_value);
}
```

Screenshots



```
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$ ./maxElementInArray
Maximum value is: 65657
axebell@akshithbellare-DELLG3:/mnt/c/Users/akshi/Desktop/year3CourseMaterials/pc301/pc301Lab/lab2$ 
```