VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE 4001 PARALLEL AND DISTRIBUTED COMPUTING

LAB ASSESSMENT 1

Name: Nakul Jadeja

Reg no.: 19BCE0660

School of Computer Science and Engineering

Course Code: CSE4001

Slot: L9+L10

Winter Semester 2021-22

Professor: Narayanamoorthy M

Question 1) Study of the following

a) Computer System Organization and Architecture

Definition: In general terms, the architecture of a computer system can be considered as a catalogue of tools or attributes that are visible to the user such as instruction sets, number of bits used for data, addressing techniques, etc.

Whereas, Organization of a computer system defines the way system is structured

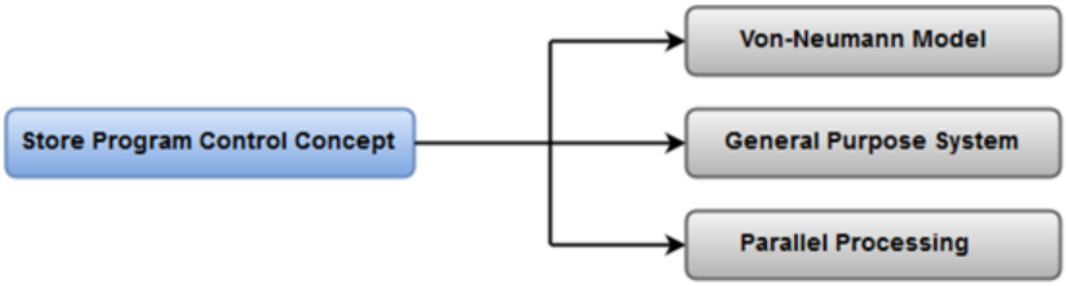
Computer Architecture	Computer Organization
Computer Architecture is concerned with the way hardware components are connected together to form a computer system.	Computer Organization is concerned with the structure and behaviour of a computer system as seen by the user.
It acts as the interface between hardware and software.	It deals with the components of a connection in a system.
Computer Architecture helps us to understand the functionalities of a system.	Computer Organization tells us how exactly all the units in the system are arranged and interconnected.
A programmer can view architecture in terms of instructions, addressing modes and registers.	Whereas Organization expresses the realization of architecture.
While designing a computer system architecture is considered first.	An organization is done on the basis of architecture.
Computer Architecture deals with high-level design issues.	Computer Organization deals with low-level design issues.
Architecture involves Logic (Instruction sets, Addressing modes, Data types, Cache optimization)	Organization involves Physical Components (Circuit design, Adders, Signals, Peripherals)

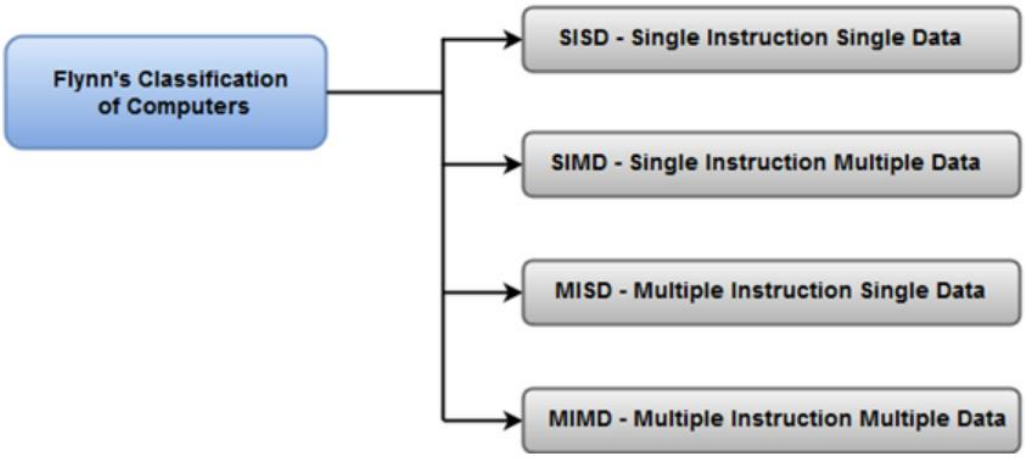
so that all those catalogued tools can be used. The significant components of Computer organization are ALU, CPU, memory and memory organization.

Computer Architecture v/s Computer Organization

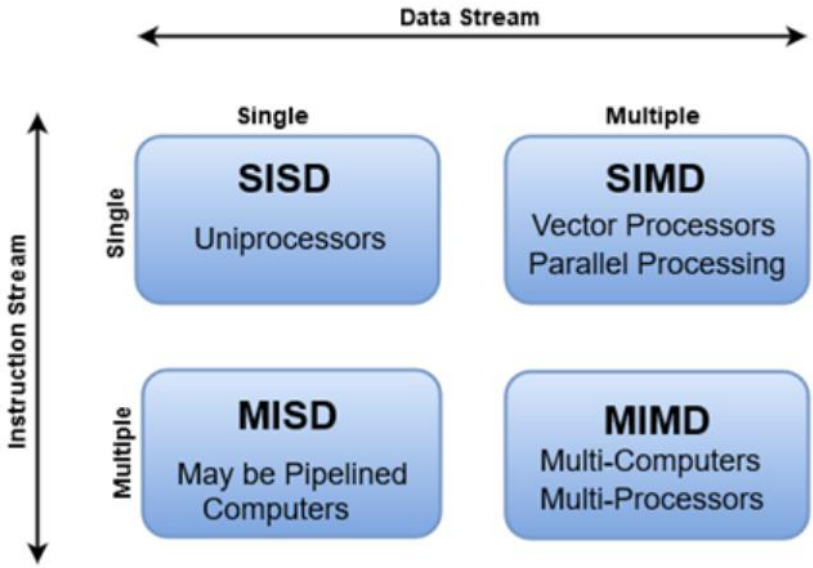
In Computer Architecture, the General System Architecture is divided into two major classification units.

1. Store Program Control Concept
2. Flynn's Classification of Computers





Flynn's Classification of Computers



b) Parallel Programming Languages and Introduction to MP:

Parallel Programming:

Parallel computing has made a tremendous impact on a variety of areas ranging from computational simulations for scientific and engineering applications to commercial applications in data mining and transaction processing. The cost benefits of parallelism coupled with the performance requirements of applications present compelling arguments in favour of parallel computing. We present a small sample of the diverse applications of parallel computing.

The traditional logical view of a sequential computer consists of a memory connected to a processor via a datapath. All three components – processor, memory, and datapath – present bottlenecks to the overall processing rate of a computer system. A number of architectural innovations over the years have addressed these bottlenecks. One of the most important innovations is multiplicity – in processing units, datapaths, and memory units. This multiplicity is either entirely hidden from the programmer, as in the case of implicit parallelism, or exposed to the programmer in different forms. In this chapter, we present an overview of important architectural concepts as they relate to parallel processing. The objective is to provide sufficient detail for programmers to be able to write efficient code on a variety of platforms. We develop cost models and abstractions for quantifying the performance of various parallel algorithms, and identify bottlenecks resulting from various programming constructs.

Parallel Programming Languages:

Parallel programming languages are languages designed to program algorithms and applications on parallel computers. Parallel processing is a great opportunity for developing high performance systems and solving large problems in many application areas. During the last few years parallel computers ranging from tens to thousands of computing elements became commercially available. They continue to gain recognition as powerful tools in scientific research, information management, and engineering applications. This trend is driven by parallel programming languages and tools that contribute to make parallel computers useful in supporting a broad range of applications. Many models and languages have been designed and implemented to allow the design and development of applications on parallel computers. Parallel programming languages (called also *concurrent languages*) allow the design of parallel algorithms as a set of concurrent actions mapped onto different computing elements. The cooperation between two or more actions can be performed in many ways according to the selected language. The design of programming languages and software tools for parallel computers is essential for wide diffusion and efficient utilization of these novel architectures. High-level languages decrease both the design time and the

execution time of parallel applications, and make it easier for new users to approach parallel computers.

Introduction to Open MP:
OpenMP is an Application Program Interface (API) that may be used to explicitly direct *multi-threaded, shared memory parallelism* in C/C++ programs. It is not intrusive on the original serial code in that the OpenMP instructions are made in pragmas interpreted by the compiler.
OpenMP uses the fork-join model of parallel execution. All OpenMP programs begin with a single master thread which executes sequentially until a parallel region is encountered, when it creates a team of parallel threads (FORK). When the team threads complete the parallel region, they synchronize and terminate, leaving only the master thread that executes sequentially (JOIN).

(2)Write a C-program using Open MP to print hello world, to print the following environment detail: number of threads, thread number, number of processors and maximum threads with sample message.

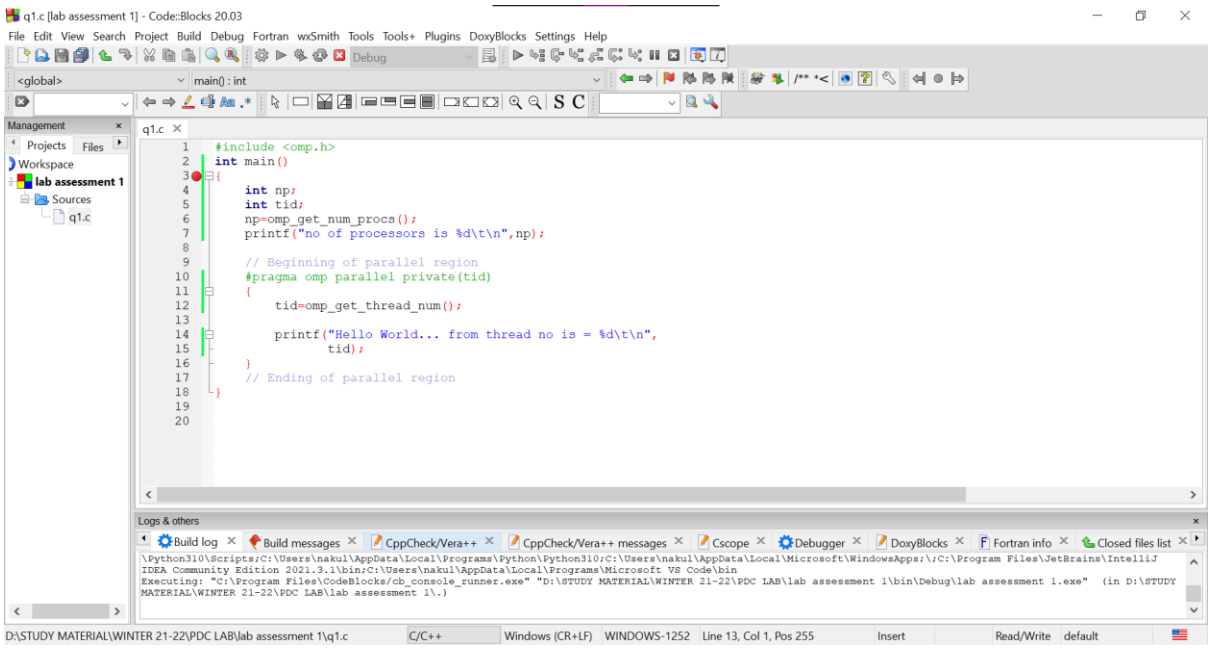
Program to print hello world:

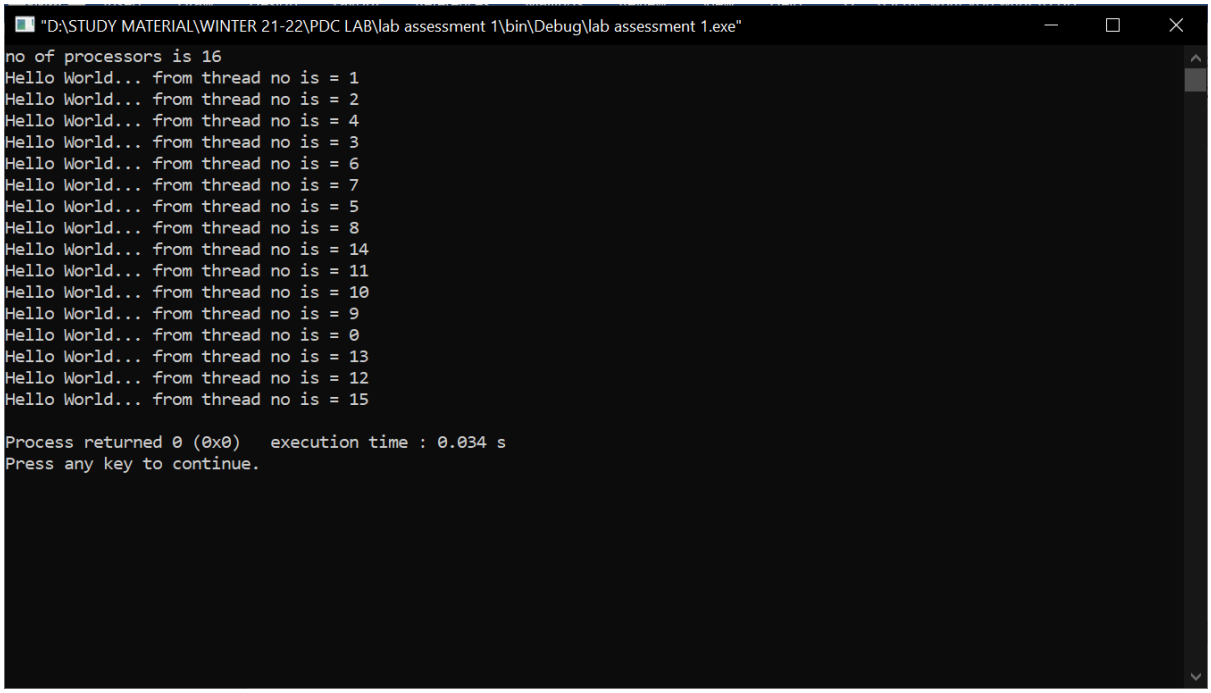
Code:

```
#include <omp.h>
int main()
{
   int np;
   int tid;
   np=omp_get_num_procs();
   printf("no of processors is %d\t\n",np);

   // Beginning of parallel region
```

```
    #pragma omp parallel private(tid)
    {
        tid=omp_get_thread_num();

        printf("Hello World... from thread no is = %d\t\n",
                tid);
    }
    // Ending of parallel region
}
```

Screenshot of the Output:





(3)Write a c-program using open MP to perform the arithmetic operations between two integers using multiple threads and measure the time?

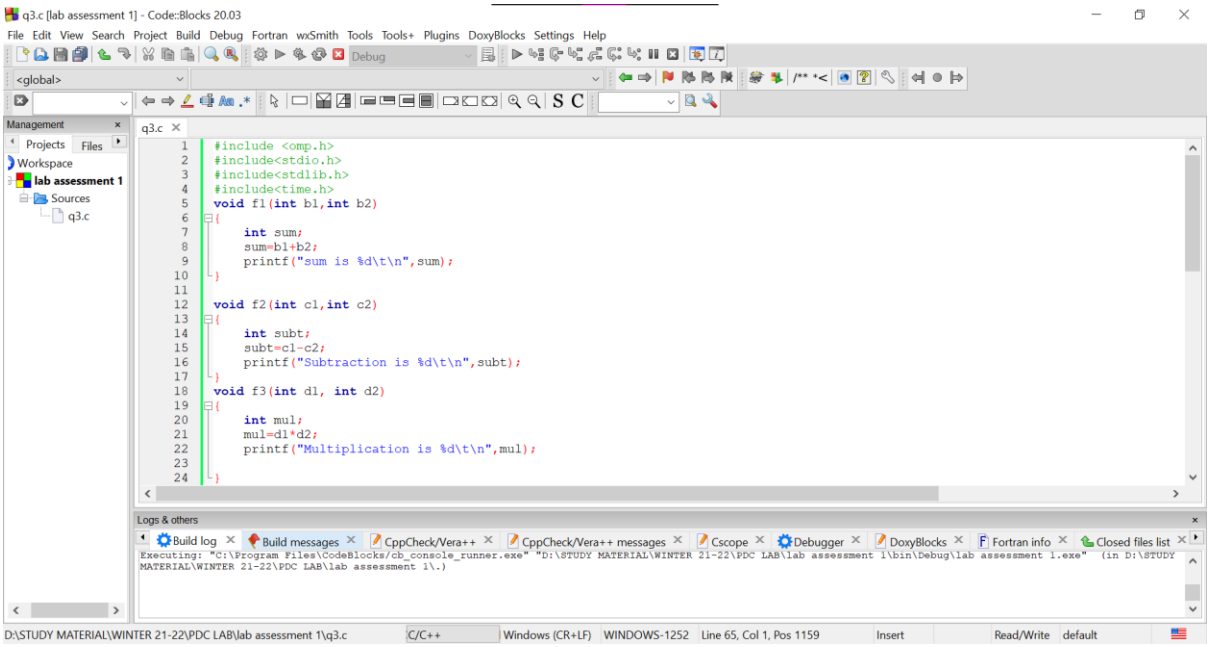
Code:

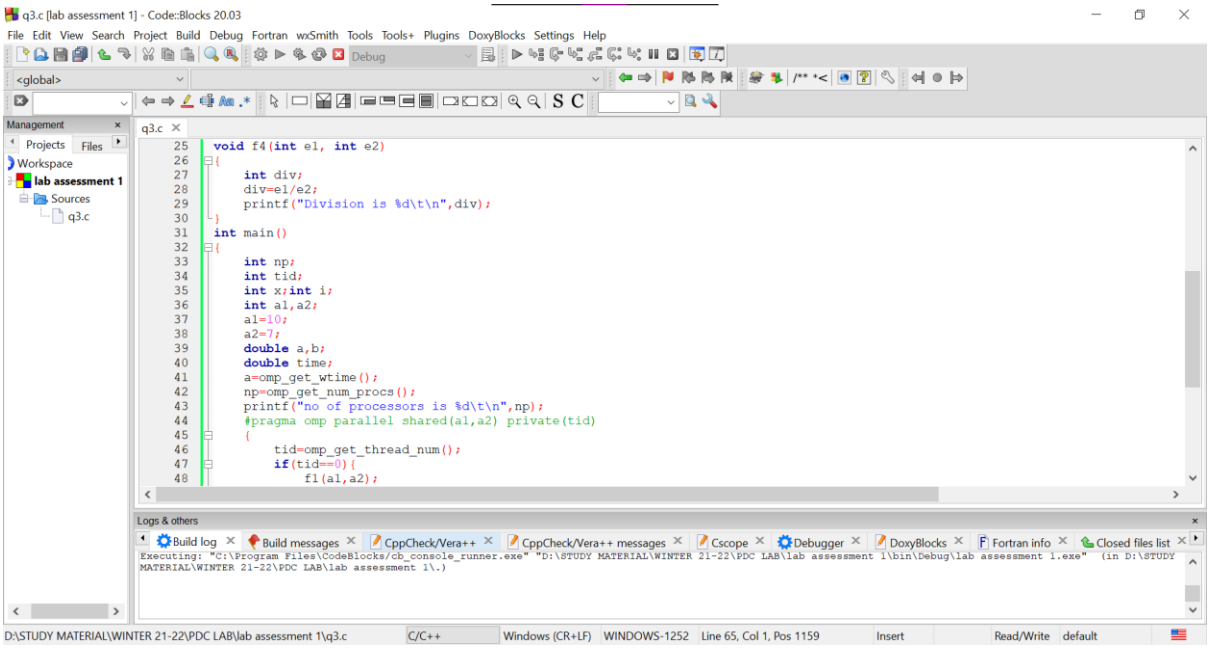
```
#include <omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void f1(int b1,int b2)
{
   int sum;
   sum=b1+b2;
   printf("sum is %d\t\n",sum);
}

void f2(int c1,int c2)
{
```

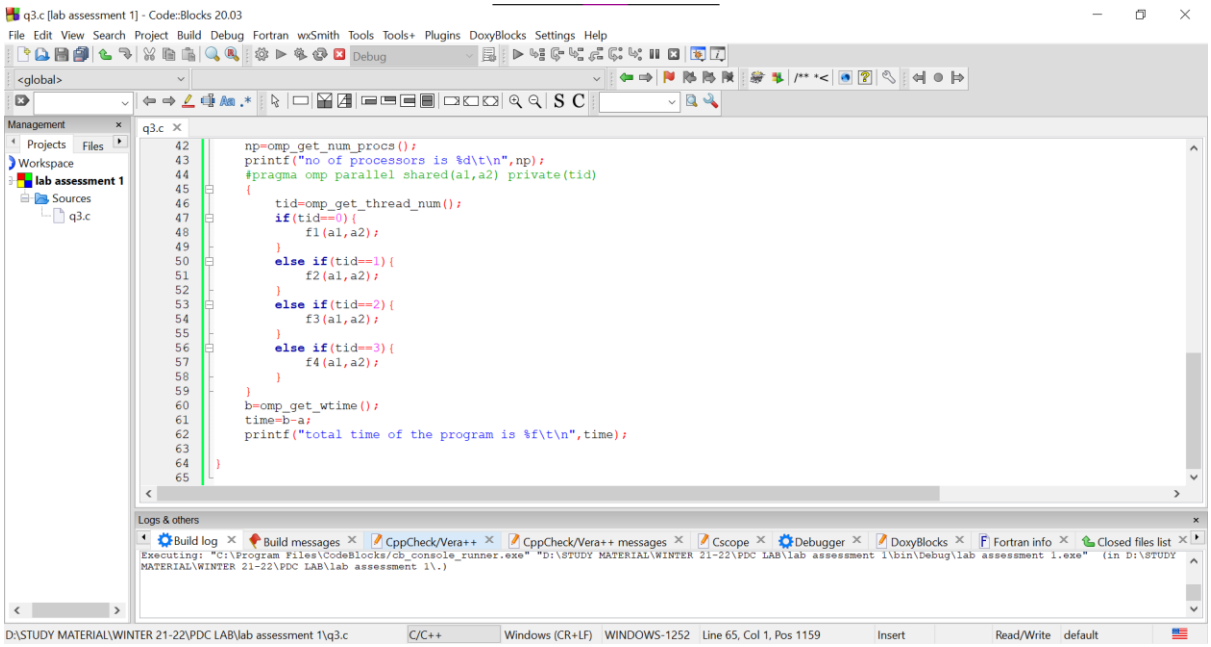
```
    int subt;
    subt=c1-c2;
    printf("Subtraction is %d\t\n",subt);
}
void f3(int d1, int d2)
{
    int mul;
    mul=d1*d2;
    printf("Multiplication is %d\t\n",mul);

}
void f4(int e1, int e2)
{
    int div;
    div=e1/e2;
    printf("Division is %d\t\n",div);
}
int main()
{
    int np;
    int tid;
    int x;int i;
    int a1,a2;
    a1=10;
    a2=7;
    double a,b;
    double time;
```

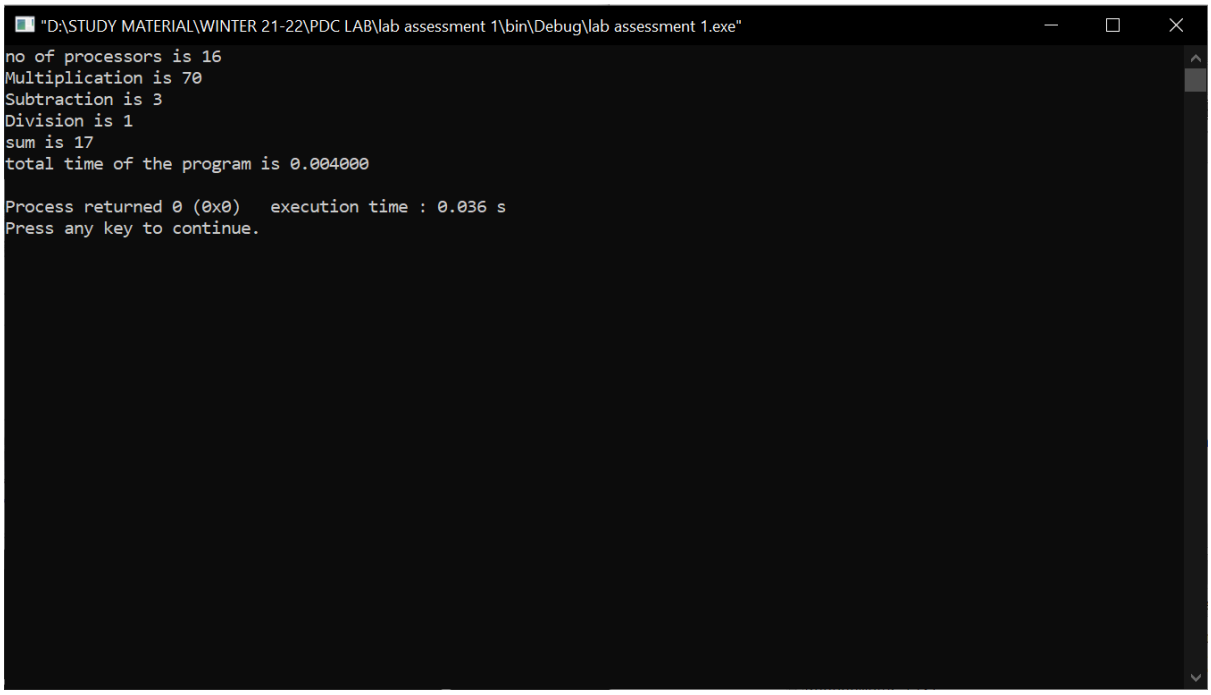
```
    a=omp_get_wtime();
    np=omp_get_num_procs();
    printf("no of processors is %d\t\n",np);
    #pragma omp parallel shared(a1,a2) private(tid)
    {
       tid=omp_get_thread_num();
       if(tid==0){
          f1(a1,a2);
       }
       else if(tid==1){
          f2(a1,a2);
       }
       else if(tid==2){
          f3(a1,a2);
       }
       else if(tid==3){
          f4(a1,a2);
       }
    }
    b=omp_get_wtime();
    time=b-a;
    printf("total time of the program is %f\t\n",time);

}
```

Screenshot of the Output:


q3.c [lab assessment 1] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Files
Workspace
lab assessment 1
Sources
q3.c
q3.c
#include <omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void f1(int b1,int b2)
{
int sum;
sum=b1+b2;
printf("sum is %d\t\n",sum);
}
void f2(int c1,int c2)
{
int subt;
subt=c1-c2;
printf("Subtraction is %d\t\n",subt);
}
void f3(int d1, int d2)
{
int mul;
mul=d1*d2;
printf("Multiplication is %d\t\n",mul);
}
Logs & others
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
Debugger
DoxyBlocks
Fortran info
Closed files list
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\bin\Debug\lab assessment 1.exe" (in D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\.)
D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\q3.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 65, Col 1, Pos 1159
Insert
Read/Write
default


q3.c [lab assessment 1] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Files
Workspace
lab assessment 1
Sources
q3.c
q3.c
void f4(int e1, int e2)
{
int div;
div=e1/e2;
printf("Division is %d\t\n",div);
}
int main()
{
int np;
int tid;
int x;int i;
int a1,a2;
a1=10;
a2=7;
double a,b;
double time;
a=omp_get_wtime();
np=omp_get_num_procs();
printf("no of processors is %d\t\n",np);
#pragma omp parallel shared(a1,a2) private(tid)
{
tid=omp_get_thread_num();
if(tid==0){
f1(a1,a2);
Logs & others
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
Debugger
DoxyBlocks
Fortran info
Closed files list
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\bin\Debug\lab assessment 1.exe" (in D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\.)
D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\q3.c
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 65, Col 1, Pos 1159
Insert
Read/Write
default





(4)Write a c-program using open MP to find the largest and smallest among

three numbers using thread approach.

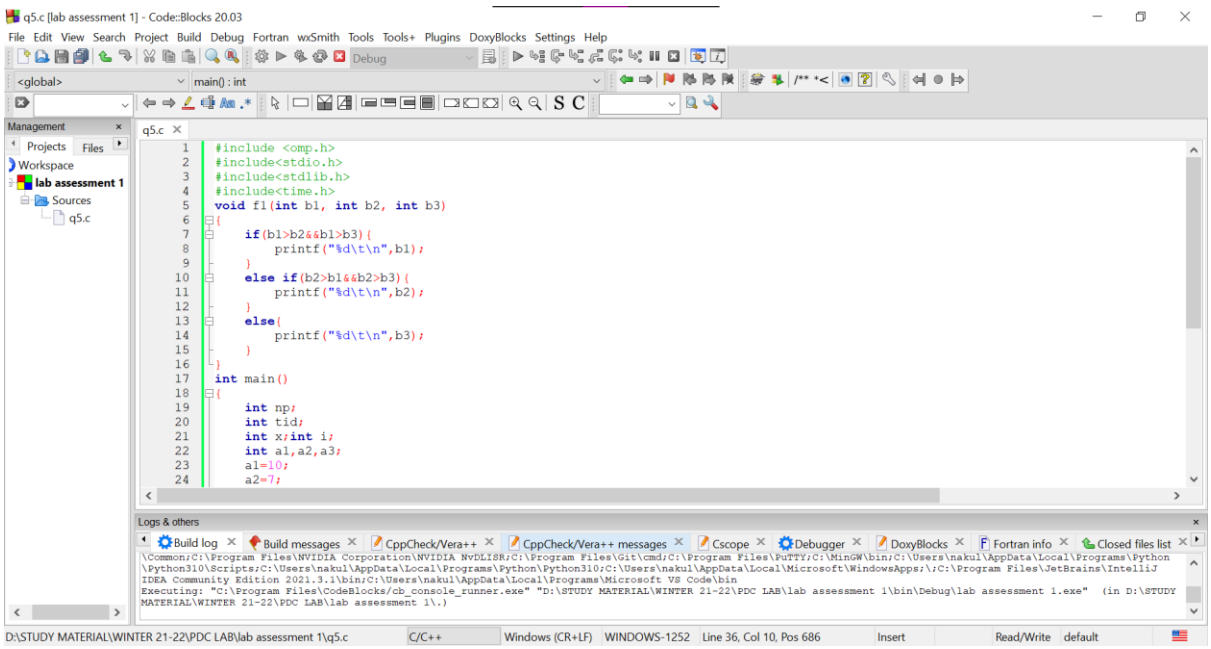
Code:

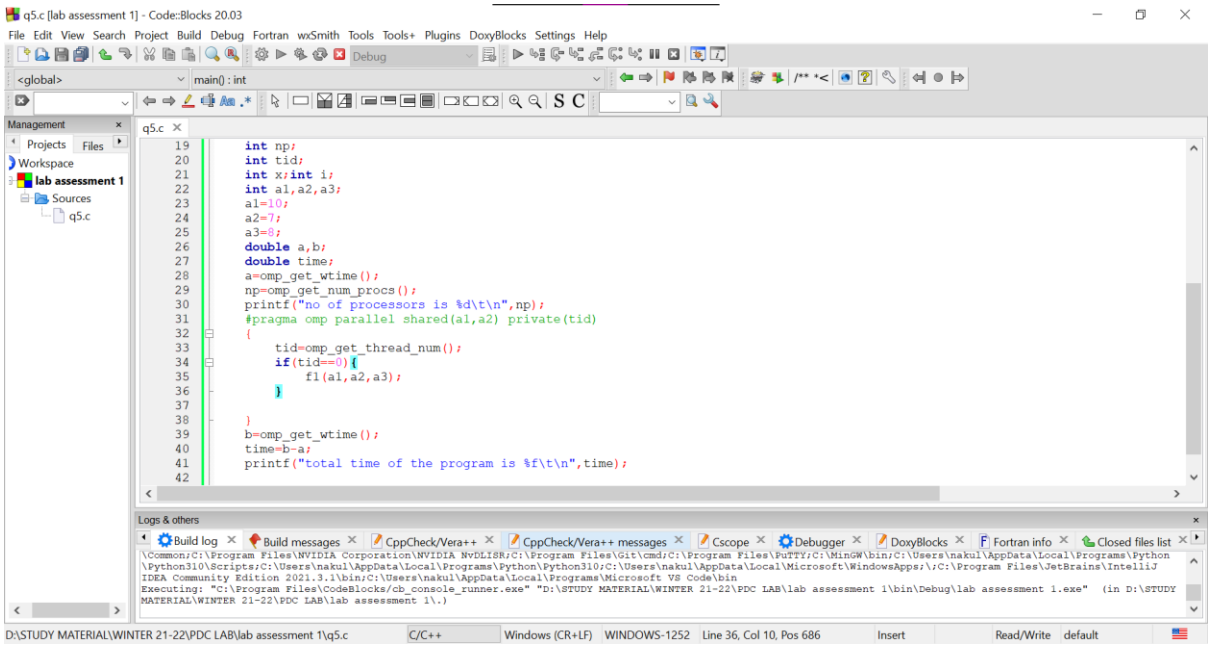
```
#include <omp.h>

#include<stdio.h>
```

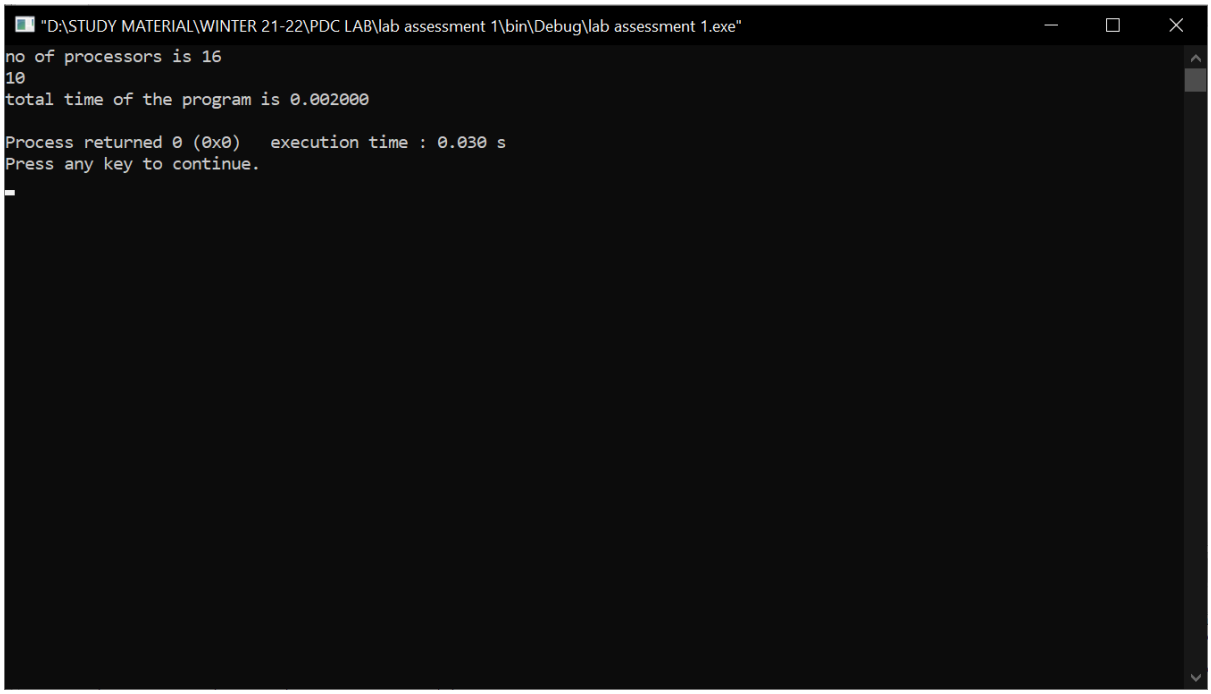
```
#include<stdlib.h>
#include<time.h>
void f1(int b1, int b2, int b3)
{
   if(b1>b2&&b1>b3){
      printf("%d\t\n",b1);
   }
   else if(b2>b1&&b2>b3){
      printf("%d\t\n",b2);
   }
   else{
      printf("%d\t\n",b3);
   }
}
int main()
{
   int np;
   int tid;
   int x;int i;
   int a1,a2,a3;
   a1=10;
   a2=7;
   a3=8;
   double a,b;
   double time;
   a=omp_get_wtime();
   np=omp_get_num_procs();
```

```
    printf("no of processors is %d\t\n",np);
    #pragma omp parallel shared(a1,a2) private(tid)
    {
       tid=omp_get_thread_num();
       if(tid==0){
          f1(a1,a2,a3);
       }

    }
    b=omp_get_wtime();
    time=b-a;
    printf("total time of the program is %f\t\n",time);

}
```

Screenshot of the Output:







[Analyse the time between serial and parallel approach]

(5)Write a C-program using Open MP to demonstrate the shared, private, first private, last private and thread private concepts

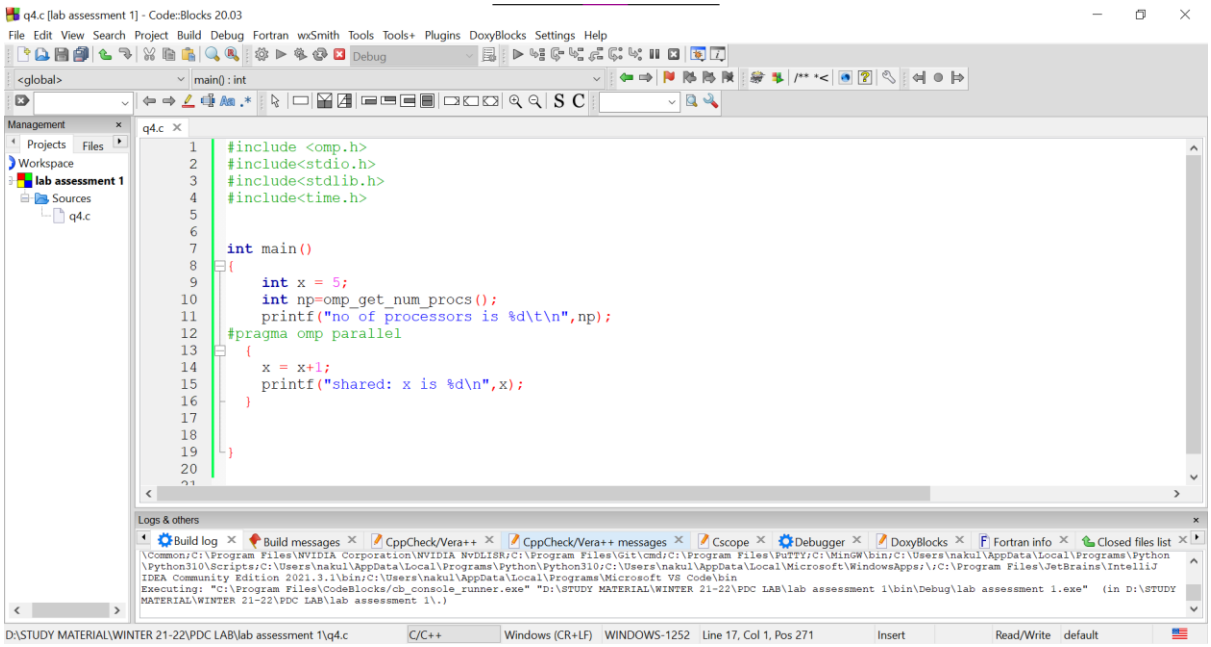
Shared Data:

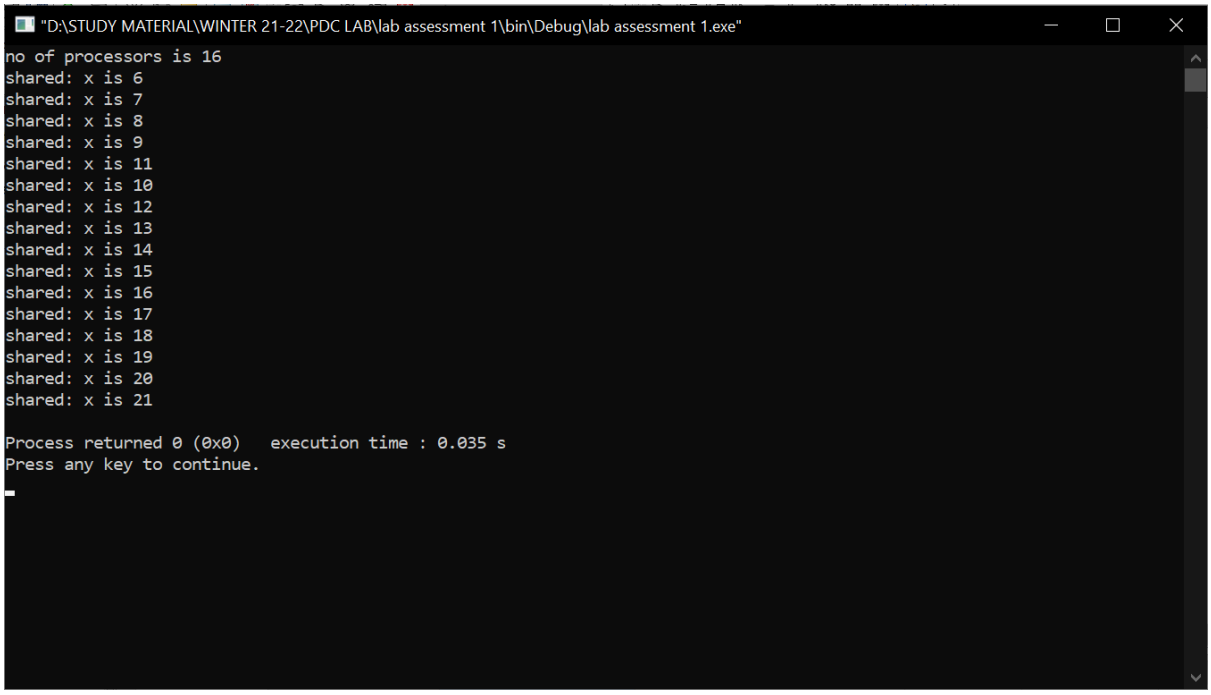
In a parallel region, any data declared outside it will be shared: any thread using a variable~ x will access the same memory location associated with that variable.

Code:

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


int main()
{
   int x = 5;
   int np=omp_get_num_procs();
   printf("no of processors is %d\t\n",np);
#pragma omp parallel
  {
   x = x+1;
   printf("shared: x is %d\n",x);
  }


}
```

Screenshot of the Output:





Private:

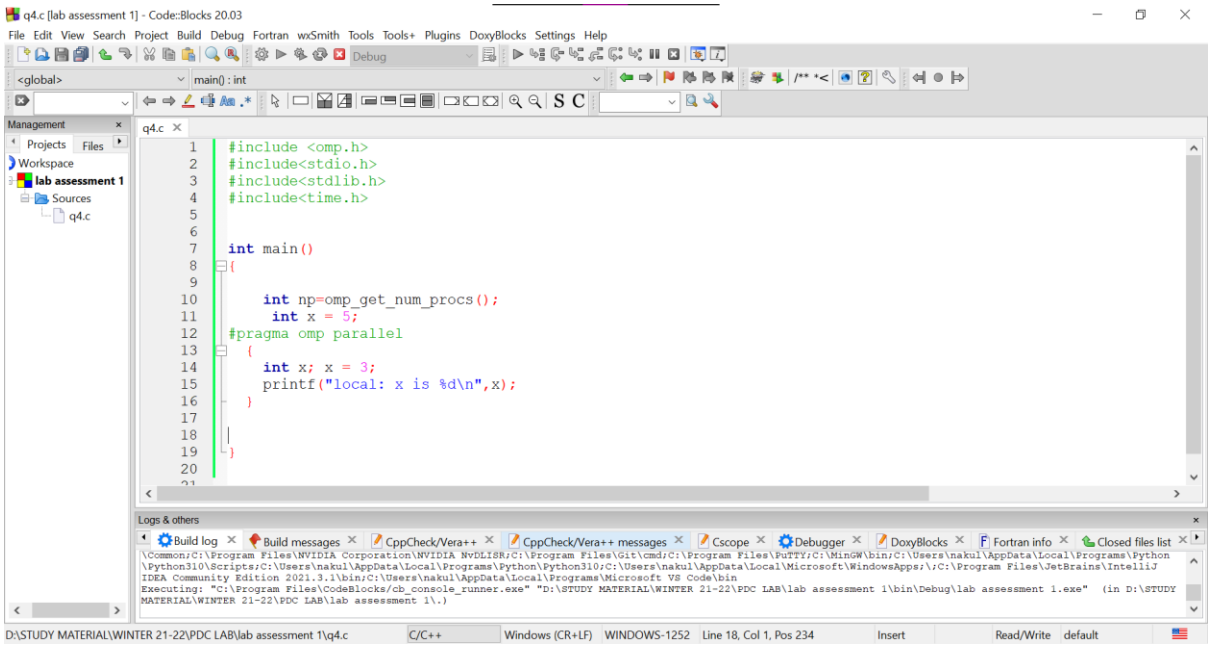
In the C/C++ language it is possible to declare variables inside a *lexical scope* ; roughly: inside curly braces. This concept extends to OpenMP parallel regions and directives: any variable declared in a block following an OpenMP directive will be local to the executing thread.

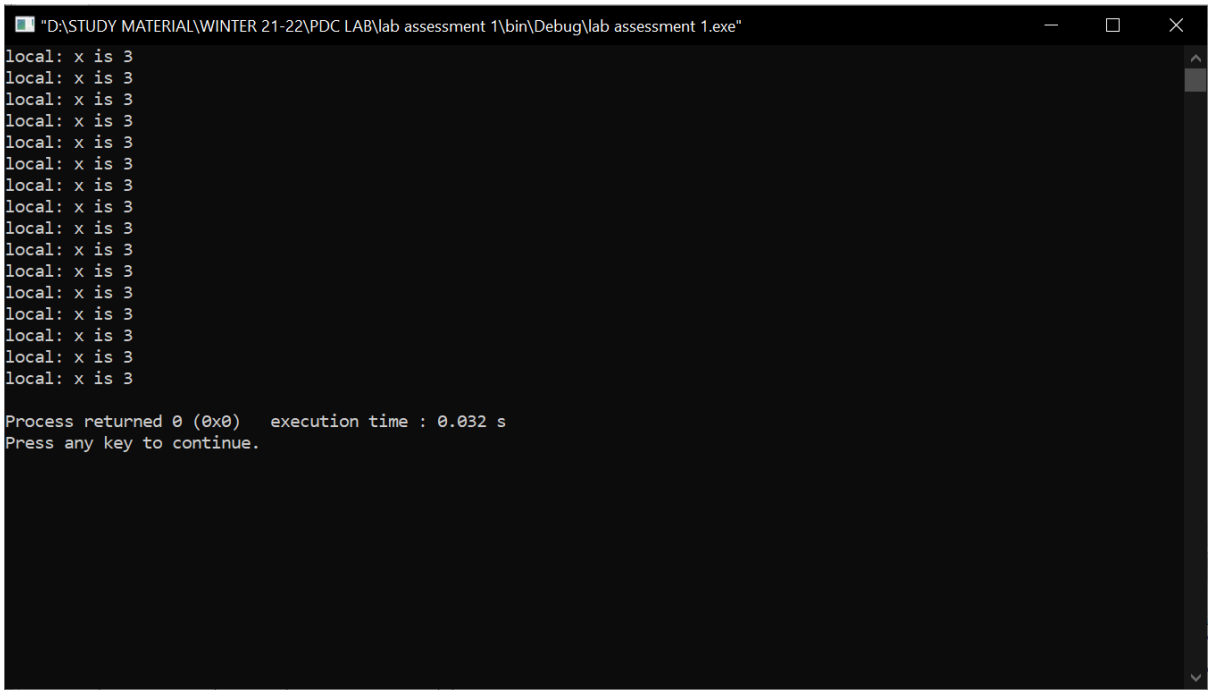
Code:

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

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


int main()
{

    int np=omp_get_num_procs();
     int x = 5;
#pragma omp parallel
  {
    int x; x = 3;
    printf("local: x is %d\n",x);
  }



}
```

Screenshot of the Output:





First Private:

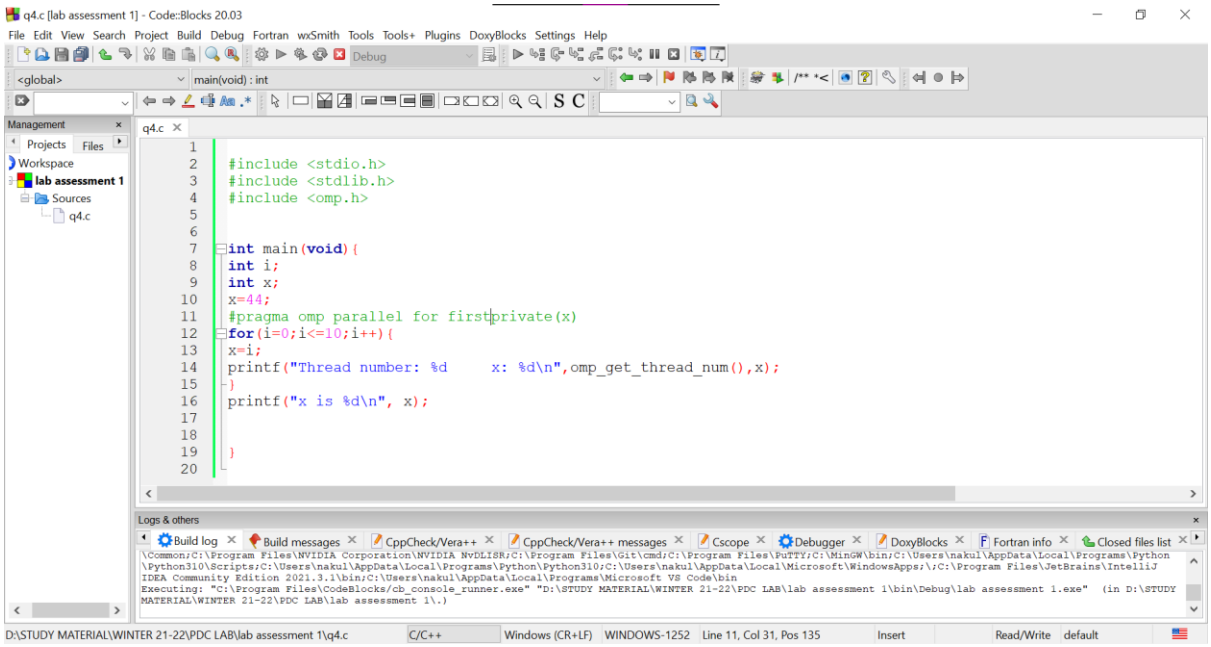
When a variable is declared as private, each thread gets a unique memory address of where to store values for that variable while in the parallel region. When the parallel region ends, the memory is freed and these variables no longer exist.

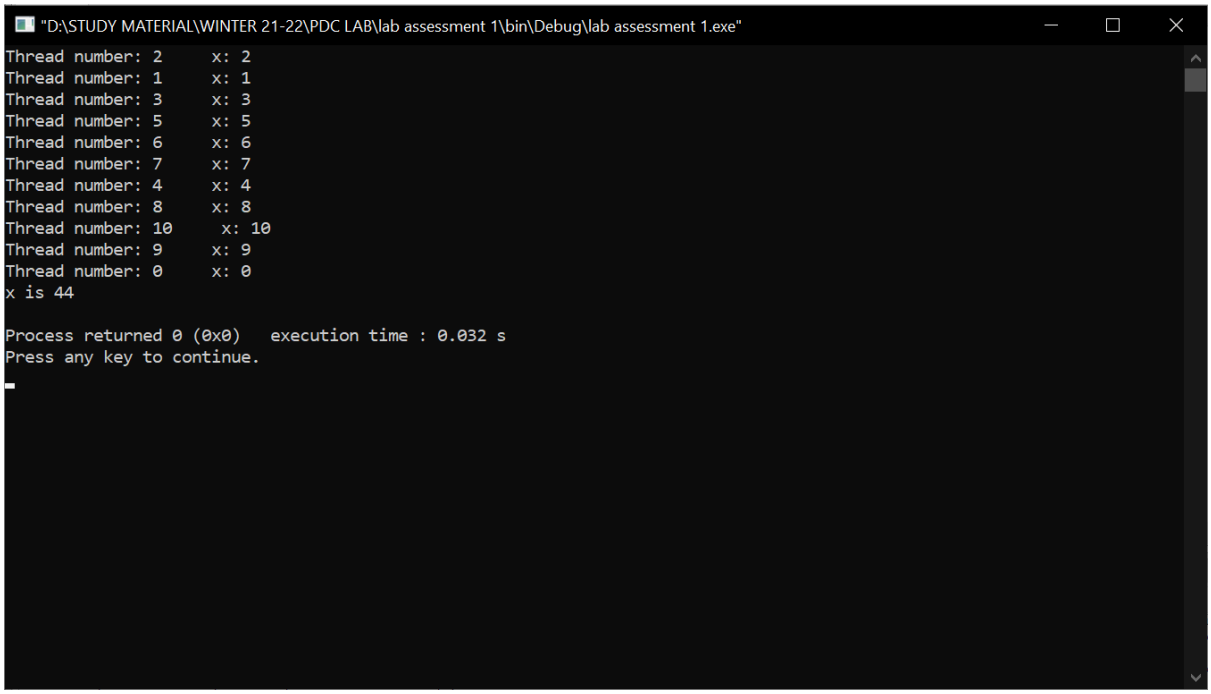
firstprivate Specifies that each thread should have its own instance of a variable, and that the variable should be initialized with the value of the variable, because it exists before the parallel construct.

Code:

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


int main(void){
int i;
int x;
x=44;
#pragma omp parallel for firstprivate(x)
for(i=0;i<=10;i++){
x=i;
printf("Thread number: %d      x: %d\n",omp_get_thread_num(),x);
}
printf("x is %d\n", x);


}
```

Screenshot of the Output:





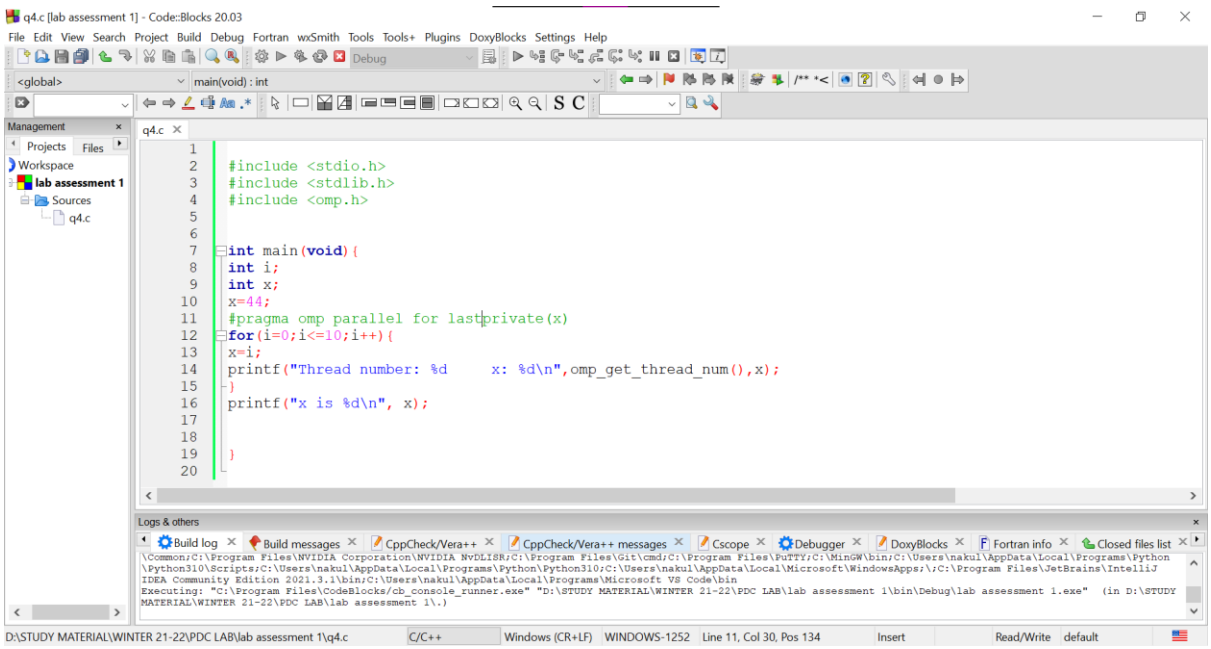
Last Private:

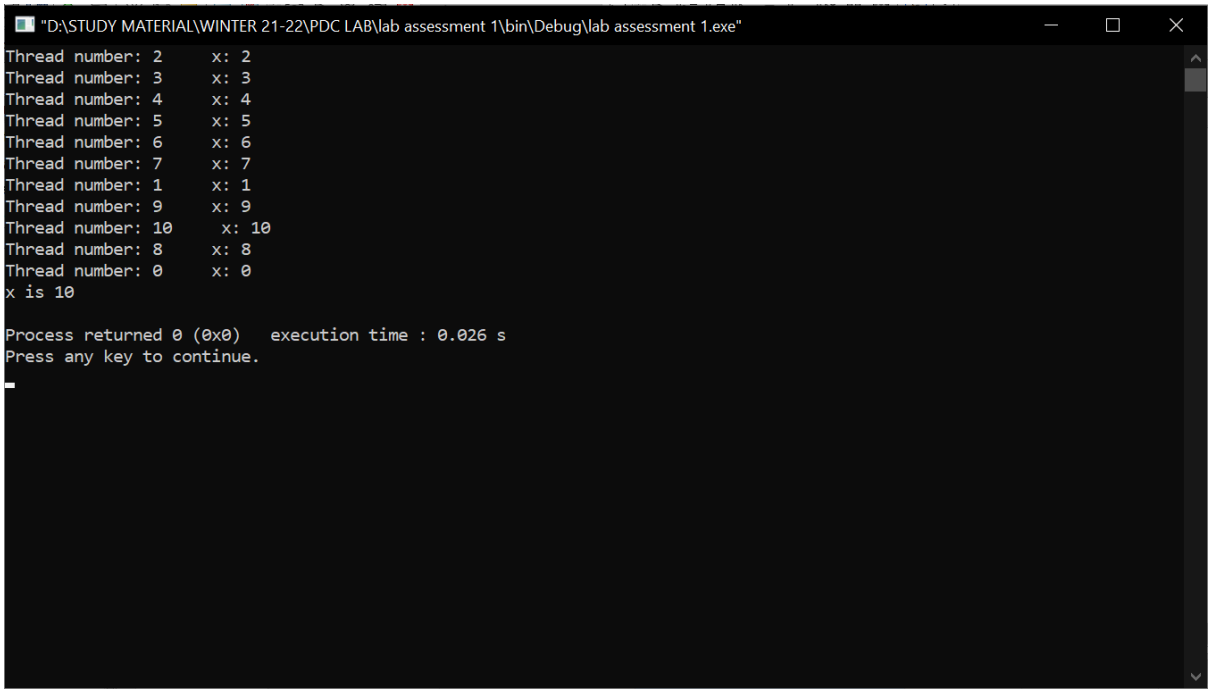
Code:

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

```
int main(void){
int i;
int x;
x=44;
#pragma omp parallel for lastprivate(x)
for(i=0;i<=10;i++){
x=i;
printf("Thread number: %d      x: %d\n",omp_get_thread_num(),x);
}
printf("x is %d\n", x);



}
```

Screenshot of the Output:




"D:\STUDY MATERIAL\WINTER 21-22\PDC LAB\lab assessment 1\bin\Debug\lab assessment 1.exe"
Thread number: 2 x: 2
Thread number: 3 x: 3
Thread number: 4 x: 4
Thread number: 5 x: 5
Thread number: 6 x: 6
Thread number: 7 x: 7
Thread number: 1 x: 1
Thread number: 9 x: 9
Thread number: 10 x: 10
Thread number: 8 x: 8
Thread number: 0 x: 0
x is 10
Process returned 0 (0x0) execution time : 0.026 s
Press any key to continue.