**Indira Gandhi Delhi Technical University**

**for Women**

**(Established by Govt. of Delhi vide Act 09 of 2012)**

**(Formerly Indira Gandhi Institute of Technology)**

**Kashmere Gate, Delhi - 110006**



**ASSIGNMENT**

**for**

**Object Oriented Programming using C++**

**MCA 104**

**Submitted To: Submitted By:**

Ms. Ankita Roopal Mittal

IT Department MCA 2nd Semester

IGDTUW 02004092019

**Object Oriented Programming using C++ Assignment 1**

**1. Single Inheritance**

#include <iostream>

using namespace std;

class base {

int x;

public:

void setx(int n) { x = n; }

void showx( ) { cout << x << "\n"; }

};

// Inherit as public

class derived : public base {

int y;

public:

void sety(int n) { y = n; }

void showy( ) { cout << y << "\n"; }

};

int main( ) {

derived ob;

ob.setx(10); // access member of base class

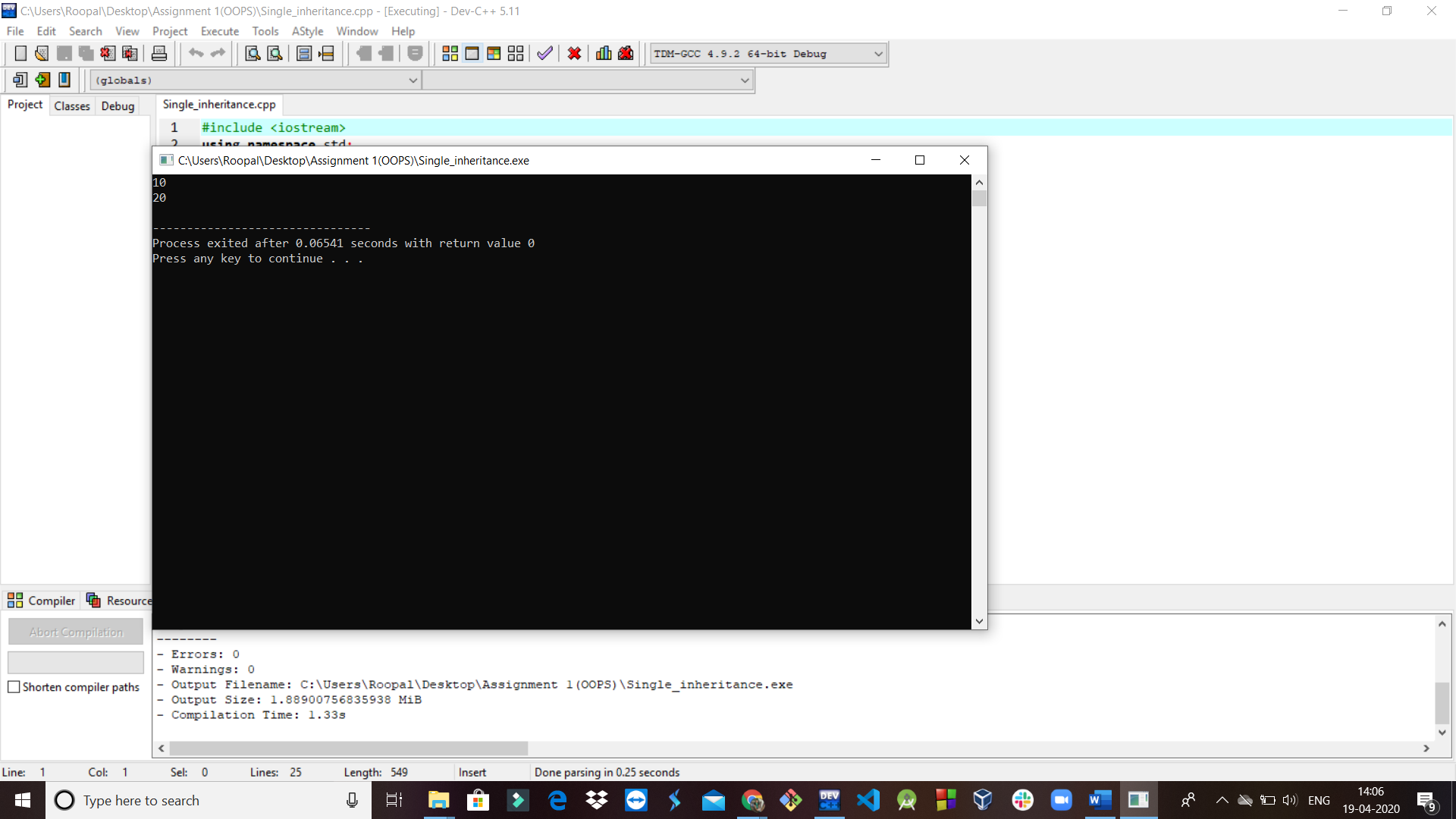
ob.sety(20); // access member of derived class

ob.showx( ); // access member of base class

ob.showy( ); // access member of derived class

return 0;

}



**2. Inheriting protected members of the base class**

#include <iostream>

using namespace std;

class base

{

protected: // private to base

int a, b; // but still accessible by derived

public:

void setab(int n, int m){a = n; b = m;}

};

// Inherit as public

class derived : public base

{

int c;

public:

void setc(int n){c = n;}

// this function has access to a and b from base

void showabc( )

{cout << a << " " << b << " " << c << "\n";}

};

int main( )

{

derived ob;

// a and b are not accessible here because they are private to both base and derived.

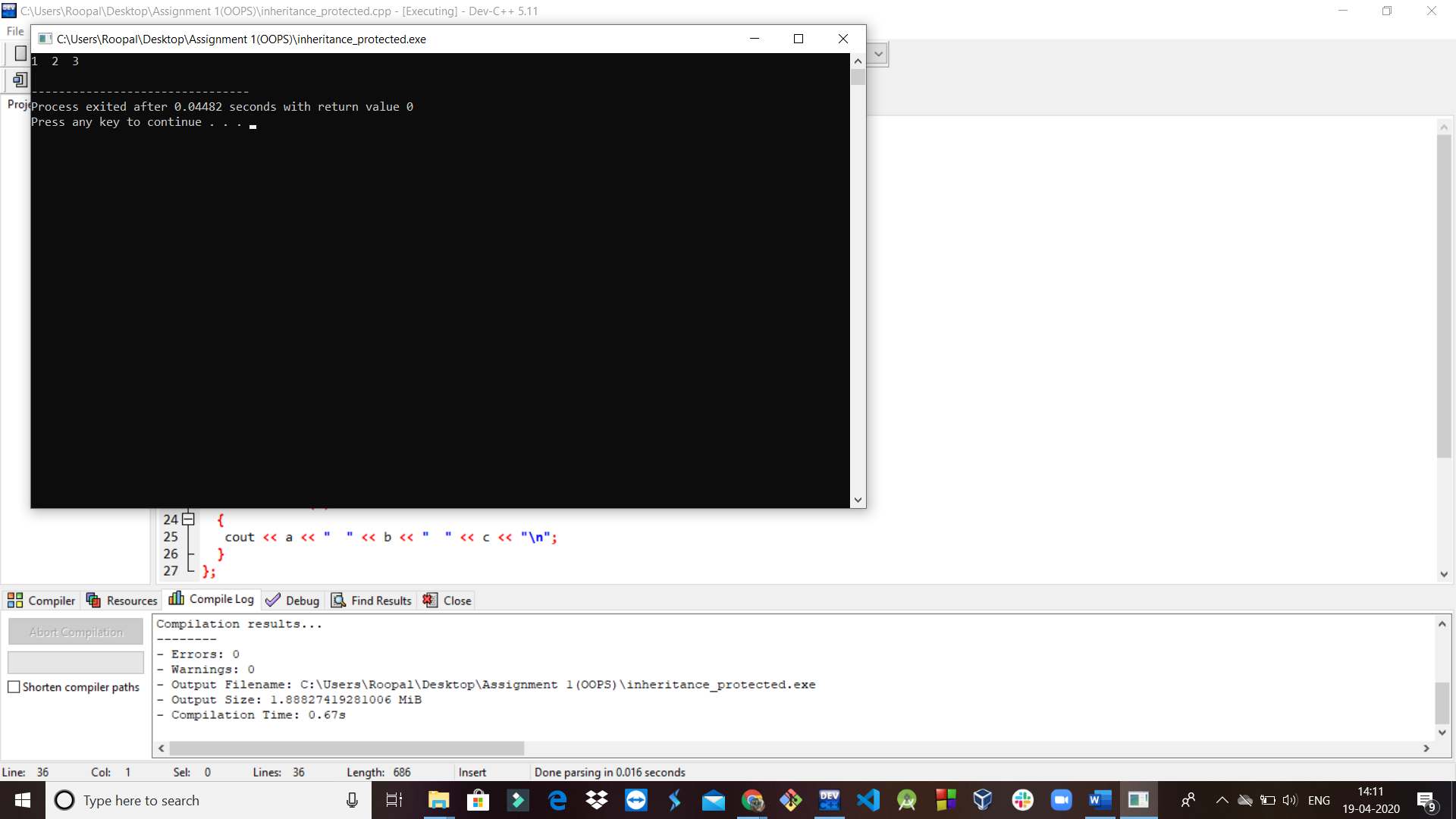
ob.setab(1, 2);

ob.setc(3);

ob.showabc( );

return 0;

}



**3. Constructor and destructor in base and derived class**

#include <iostream>

using namespace std;

class base

{

public:

base() {

cout << "Constructing base\n";

}

~base() {

cout << "Destructing base\n";

}

};

class derived : public base

{

public:

derived(){

cout << "Constructing derived\n";

}

~derived(){

cout << "Destructing derived\n";

}

};

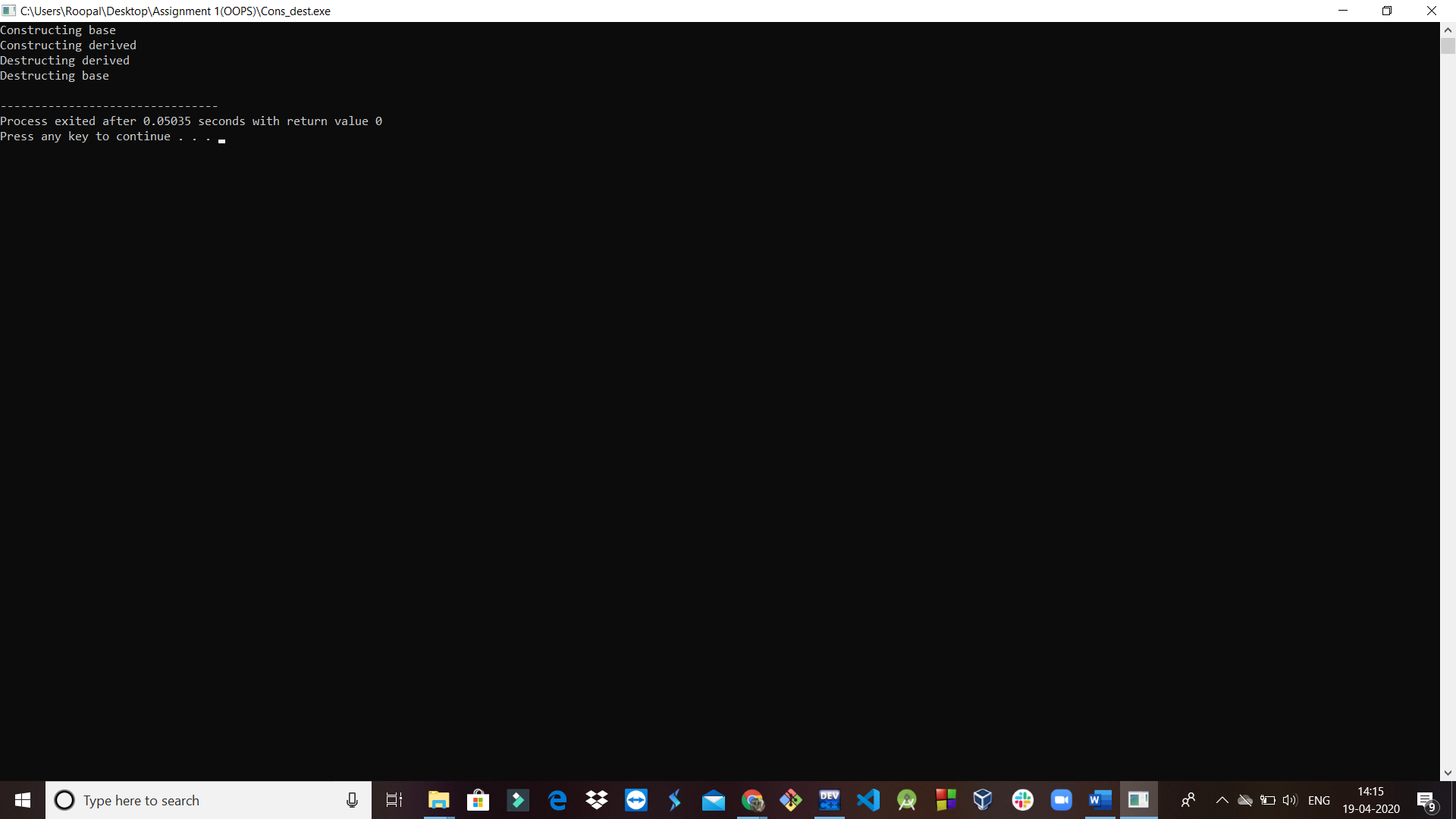
int main()

{

derived obj;

return 0;

}



**4. Constructor and destructor in base and derived class**

#include <iostream>

using namespace std;

class base

{

int i;

public:

base(int n){

cout << "Constructing base\n";

i = n;

}

~base(){

cout << "Destructing base\n";

}

void showi(){

cout << i << "\n";

}

};

class derived : public base

{

int j;

public:

derived(int n) : base(n){

// pass argument to the base class

cout << "Constructing derived\n";

j = n;

}

~derived(){

cout << "Destructing derived\n";

}

void showj(){

cout << j << "\n";

}

};

int main()

{

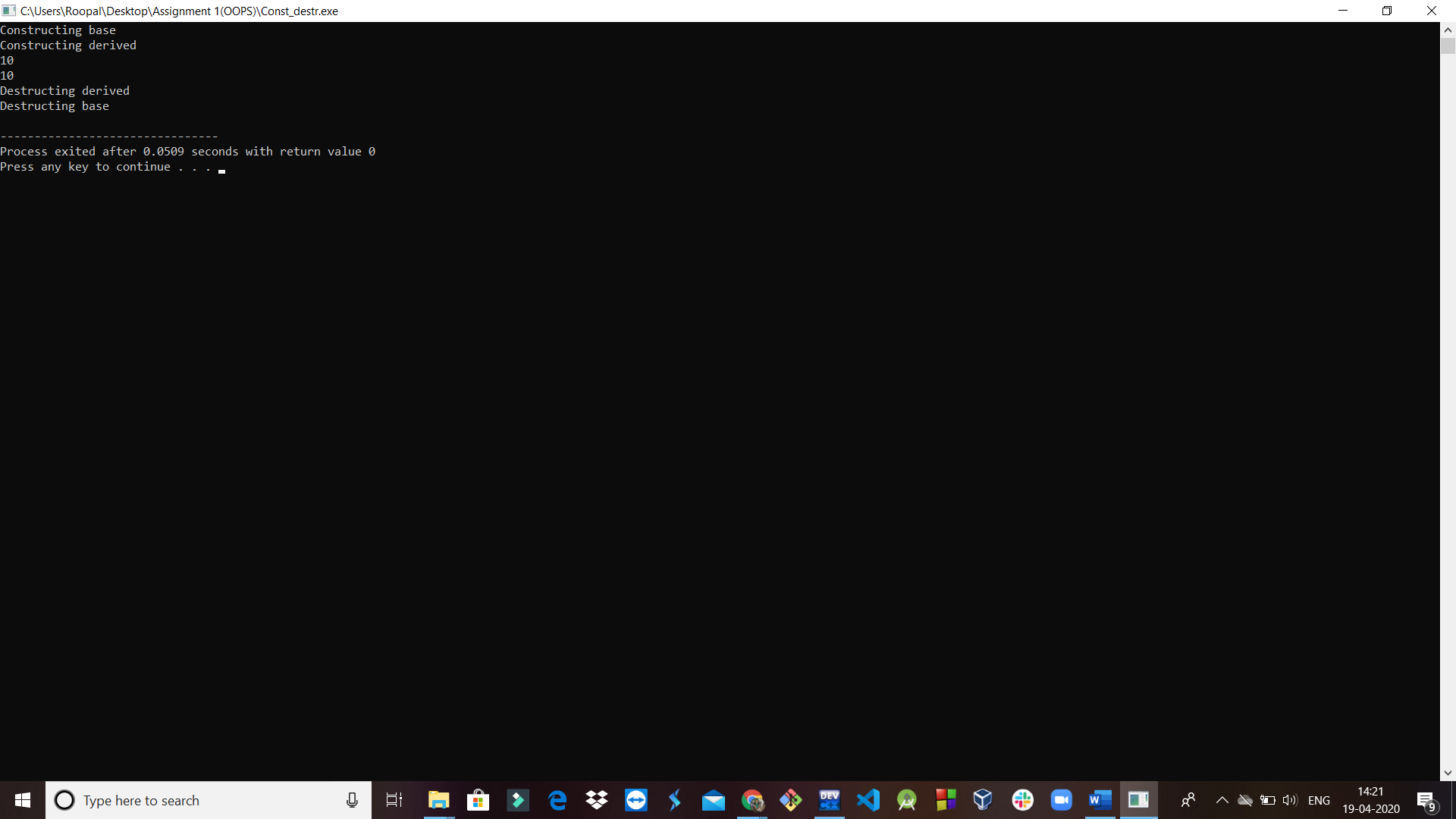
derived o(10);

o.showi();

o.showj();

return 0;

}



**5. Multiple Inheritance**

#include<iostream>

using namespace std;

class M

{

protected:

int m;

public:

void get\_m(int x){

m=x;

}

};

class N

{

protected:

int n;

public:

void get\_n(int y){

n=y;

}

};

class P

{

protected:

int m;

int n;

public:

void get\_m(int x){

m=x;

}

void get\_n(int u){

n=u;

}

void display();

};

void P :: display(){

cout<<"m = "<<m<<"\n";

cout<<"n = "<<n<<"\n";

cout<<"m\*n = "<<m\*n<<"\n";

}

int main()

{

P p;

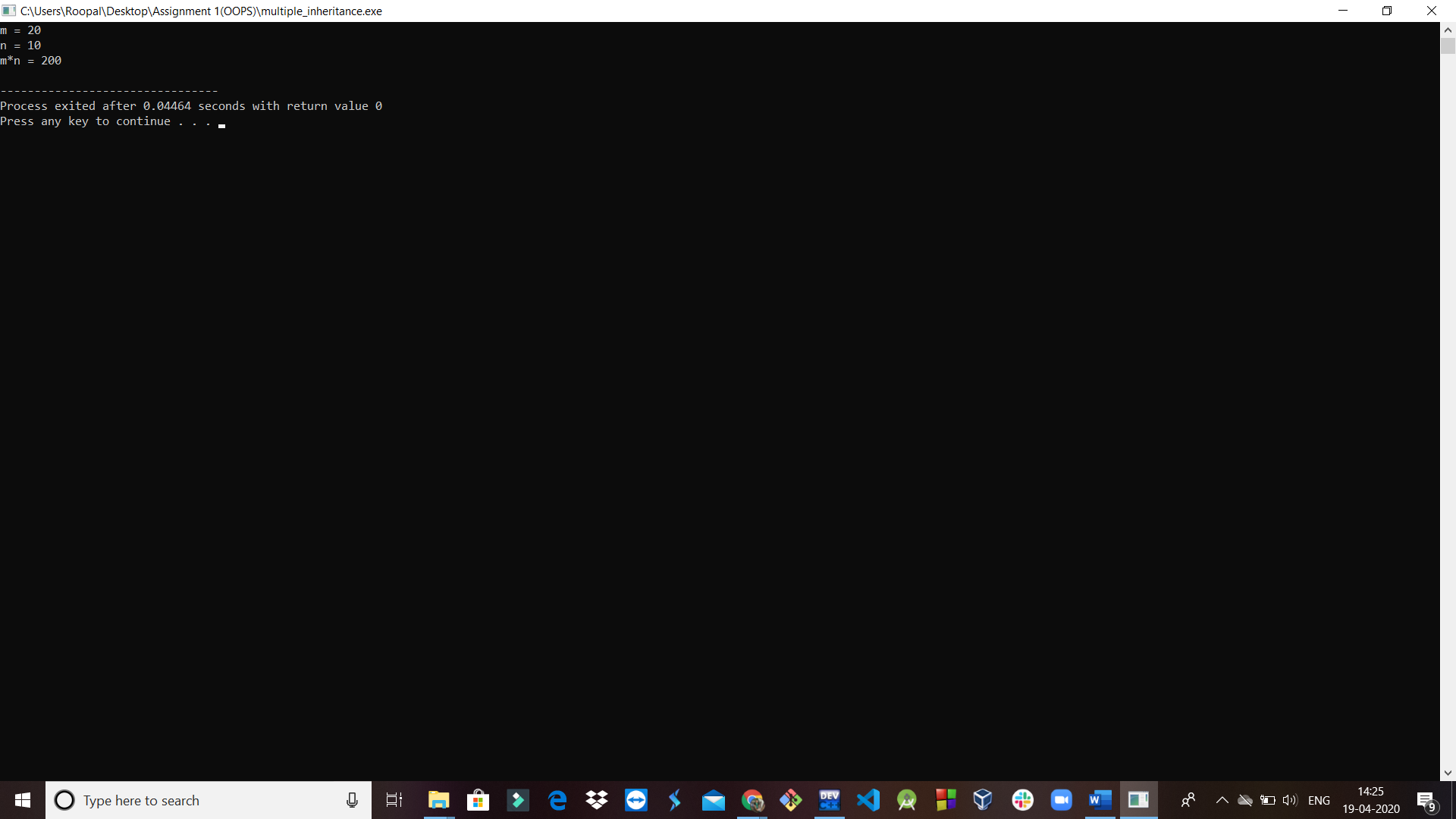
p.get\_m(20);

p.get\_n(10);

p.display();

return 0;

}



**1(c). Multilevel inheritance**

#include <iostream>

using namespace std;

class base //single base class

{

public:

int x;

void getdata()

{

cout << "Enter value of x= "; cin >> x;

}

};

class derive1 : public base // derived class from base class

{

public:

int y;

void readdata()

{

cout << "\nEnter value of y= "; cin >> y;

}

};

class derive2 : public derive1 // derived from class derive1

{

private:

int z;

public:

void indata()

{

cout << "\nEnter value of z= "; cin >> z;

}

void product()

{

cout << "\nProduct= " << x \* y \* z;

}

};

int main()

{

derive2 a; //object of derived class

a.getdata();

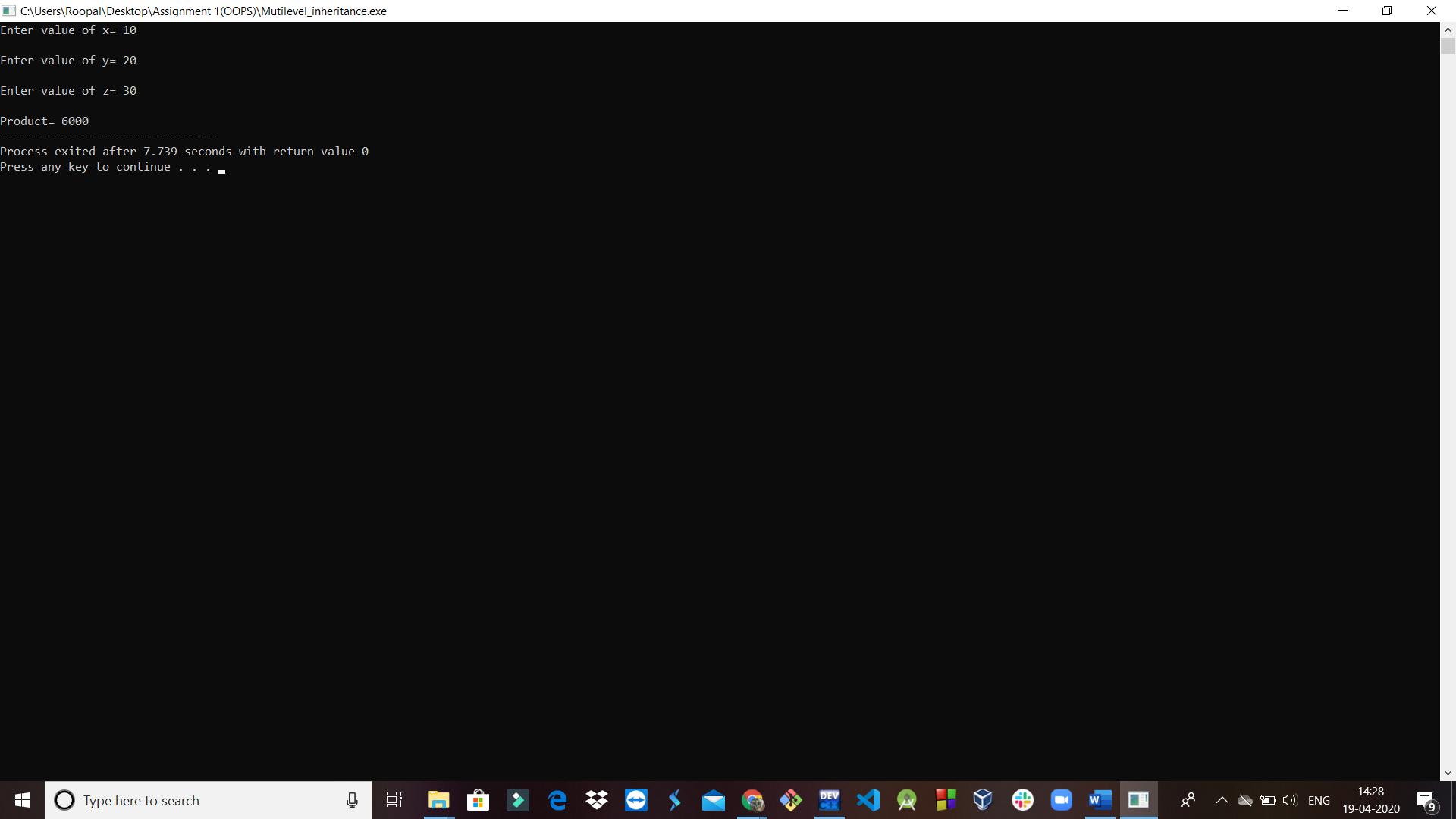
a.readdata();

a.indata();

a.product();

return 0;

}



**1(d). Ambiguity in single inheritance**

#include<iostream>

using namespace std;

class A

{

public:

void display()

{

cout<<"A\n";

}

};

class B: public A

{

public:

void display()

{

cout<<"B\n";

}

};

int main()

{

B b; //derived class object

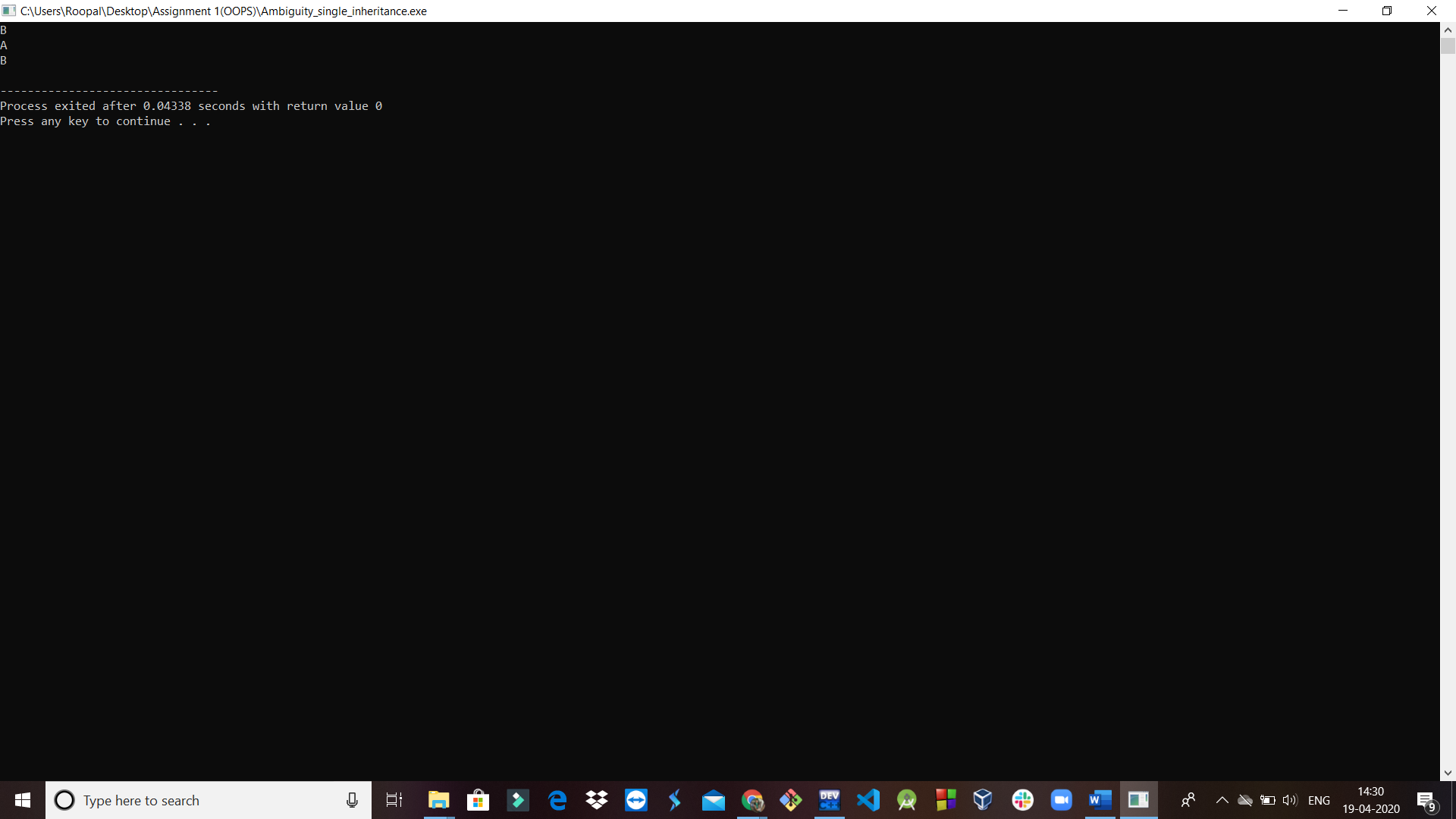
b.display(); //invokes display() in B

b.A::display(); //invokes display() in A

b.B::display(); //invokes display() in B

return 0;

}



**1(e). Ambiguity in multiple inheritance**

#include<iostream>

using namespace std;

class M

{

public:

void display(void)

{

cout<<"Class M\n";

}

};

class N

{

public:

void display(void)

{

cout<<"Class N\n";

}

};

class P: public M, public N

{

public:

void display(void) //overrides display() of M and N

{

M::display();

}

};

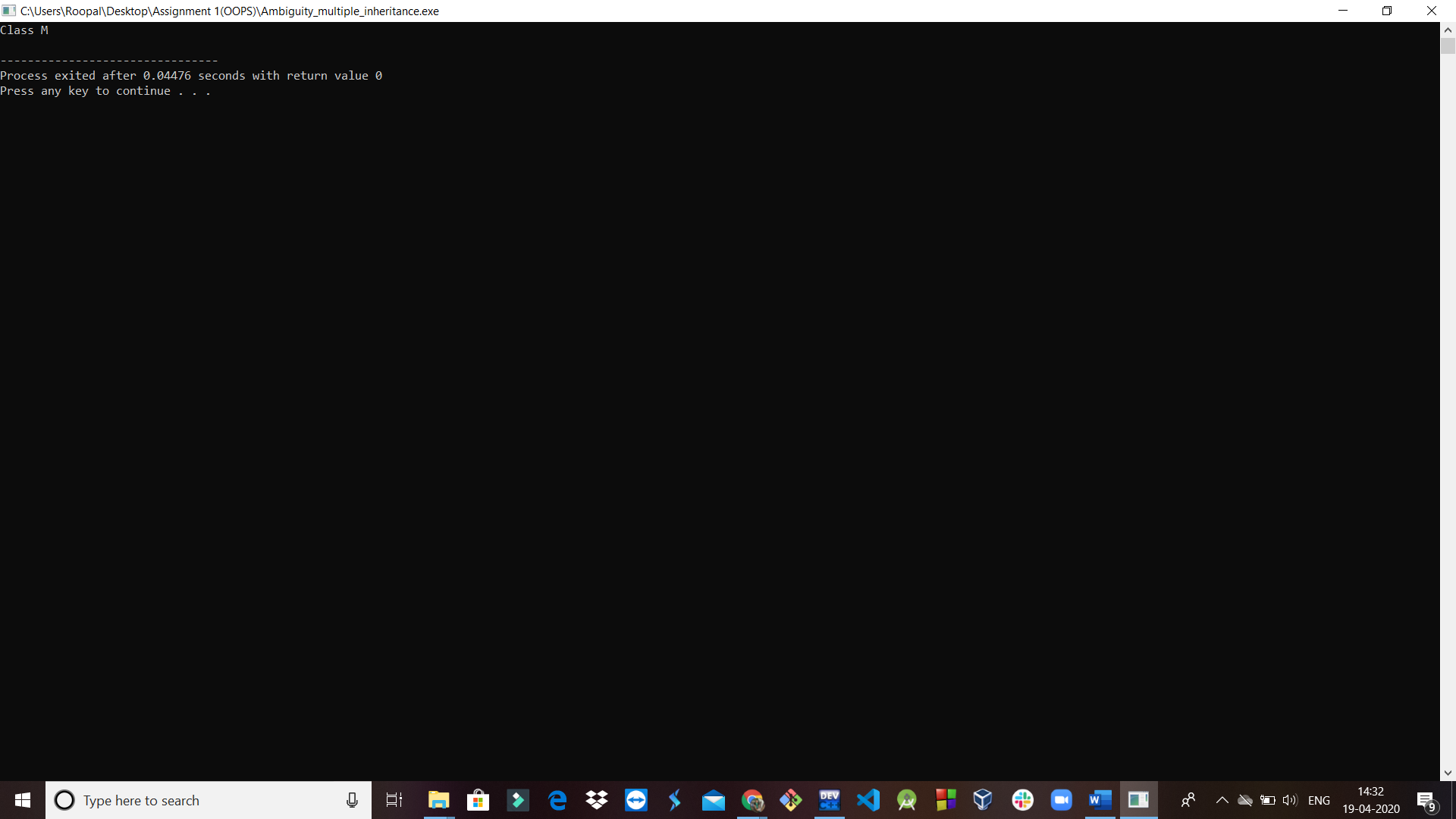
int main()

{

P p;

p.display();

}



**2. Virtual function and abstract class**

#include<iostream>

using namespace std;

class base

{

public:

int i;

base(int x){

i = x;

}

virtual void func( ){

cout << "Using base version of func(): ";

cout << i << "\n";

}

};

class derived1 : public base

{

public:

derived1(int x) : base(x) { }

void func( )

{

cout << "Using derived1's version of func(): ";

cout << i\*i << "\n";

}

};

class derived2 : public base

{

public:

derived2(int x) : base(x) { }

void func( )

{

cout << "Using derived2's version of func(): ";

cout << i+i << "\n";

}

};

int main( )

{

base \*p;

base ob(10);

derived1 d\_ob1(10);

derived2 d\_ob2(10);

p = &ob;

p -> func( ); // use base's func( )

p = &d\_ob1;

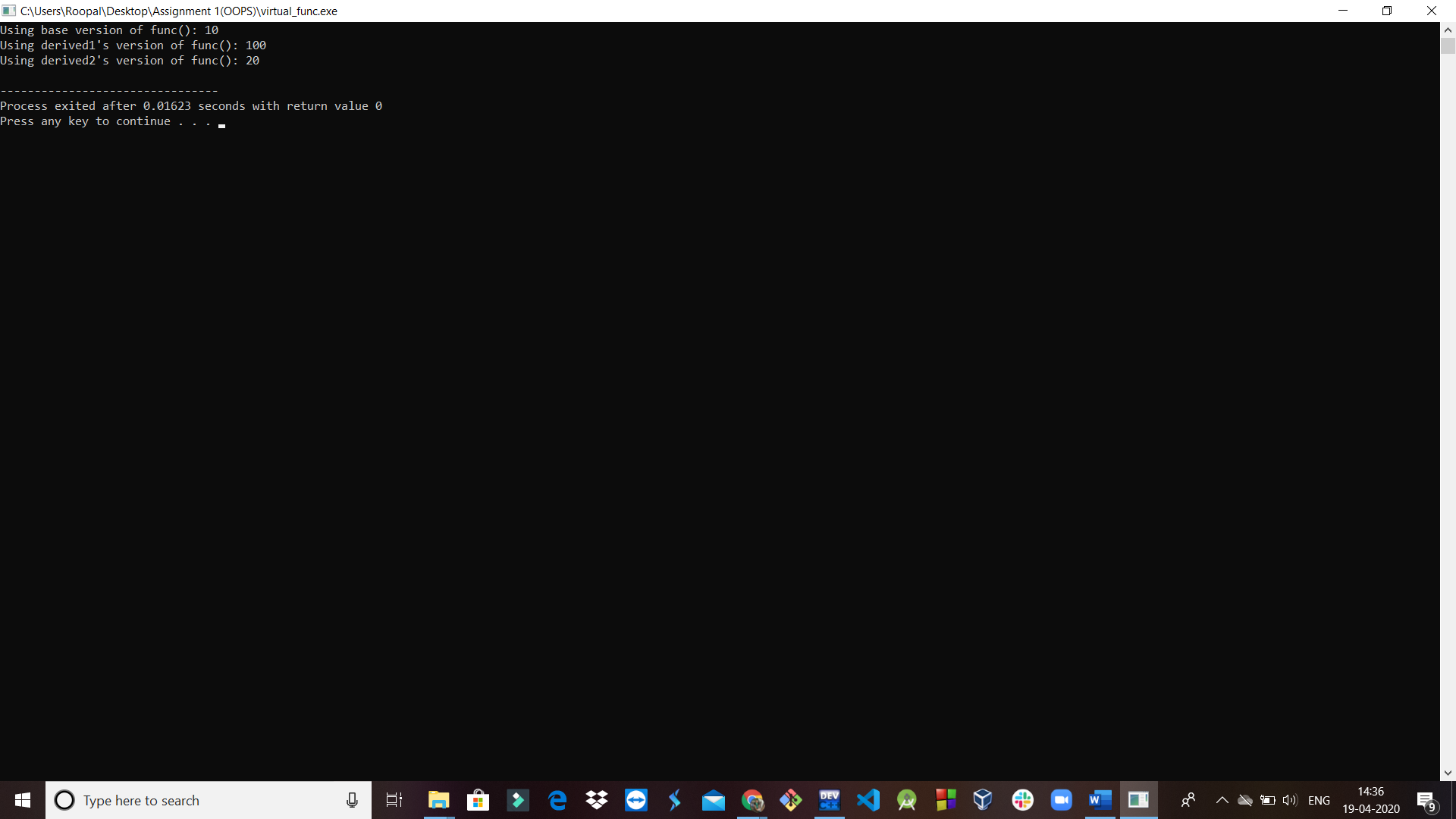
p ->func( );

p = &d\_ob2;

p ->func( ); // use derived2's func( )

return 0;

}



**3. Virtual base class**

#include<iostream>

using namespace std;

class Base

{

public:

int i;

};

// Inherit Base as virtual

class Derived1 : virtual public Base

{

public:

int j;

};

class Derived2 : virtual public Base

{

public:

int k;

};

class Derived3 : public Derived1, public Derived2

{

public:

int product()

{

return i\*j\*k;

}

};

int main()

{

Derived3 ob;

ob.i = 10; // unambiguous because virtual Base

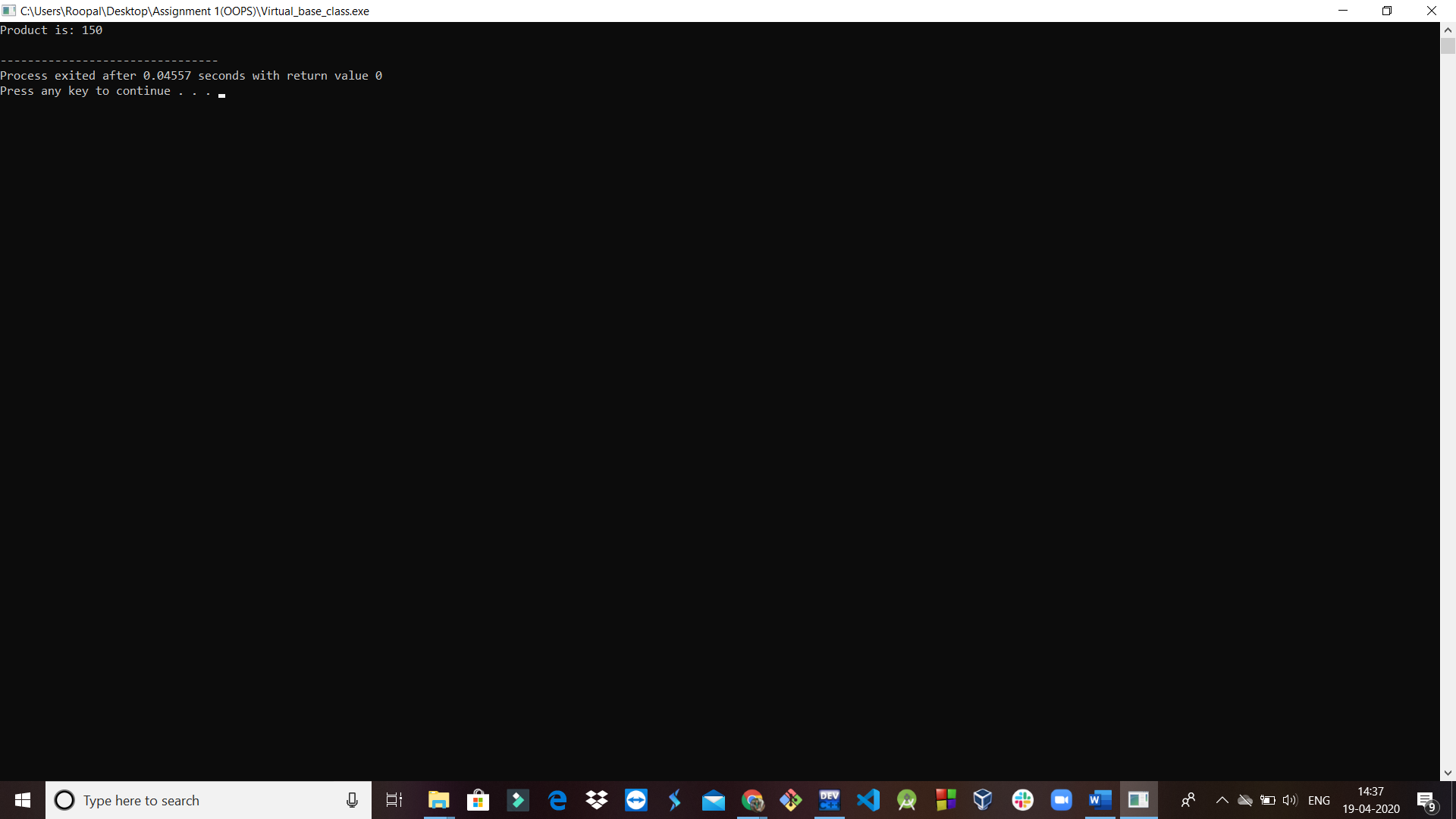
ob.j = 3;

ob.k = 5;

cout << "Product is: " << ob.product( ) << "\n";

return 0;

}



**4. Exceptional handling: Handling mechanism**

**4(a). Try**

#include<iostream>

using namespace std;

int main()

{

cout<<"Start\n";

try //start a try block

{

cout<<"Inside try block\n";

throw 10; //throw an error

cout<<"This will not execute\n";

}

catch(int i)

{

cout<<"Caught one! Number is: ";

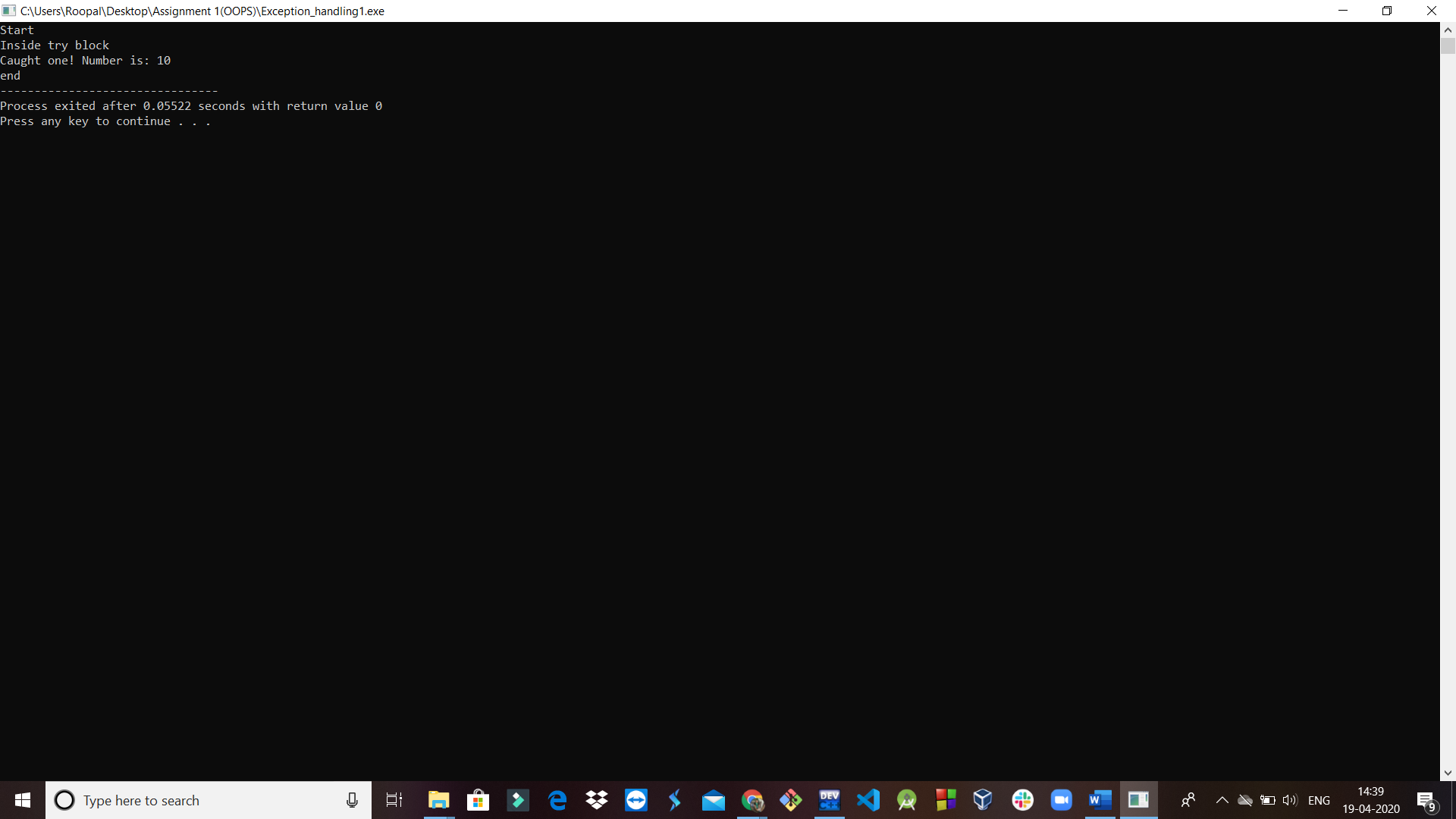
cout<<i<<"\n";

}

cout<<"end";

return 0;

}



**4(b). Catch**

#include<iostream>

using namespace std;

int main()

{

int a,b;

cout<<"Enter values";

cin>>a;

cin>>b;

int x=a-b;

try{

if(x!=0){

cout<<"result(a/x)"<<a/x<<endl;

}

else{

throw(x);

}

}

catch(int i)

{

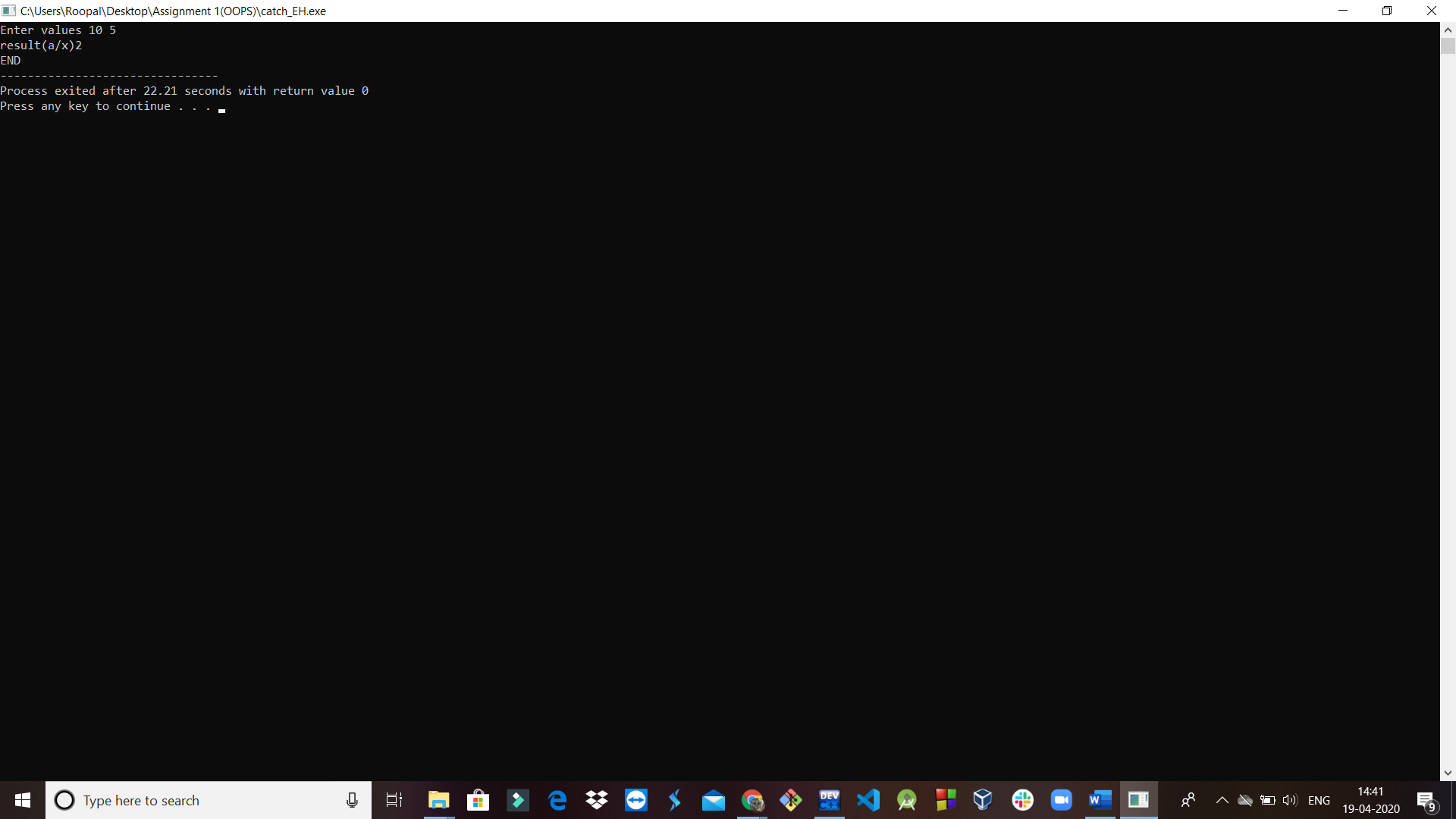
cout<<"Exception caught";

}

cout<<"END";

return 0;

}



**4(c). Throw**

#include<iostream>

using namespace std;

void Xtest(int test)

{

cout << "Inside Xtest, test is: " << test << "\n";

if (test)

throw test;

}

int main()

{

cout << "start\n";

try{

// start a try block

cout << "Inside try block\n";

Xtest(0);

Xtest(1);

Xtest(2);

}

catch (int i){

// catch an error

cout << "Caught one! Number is: ";

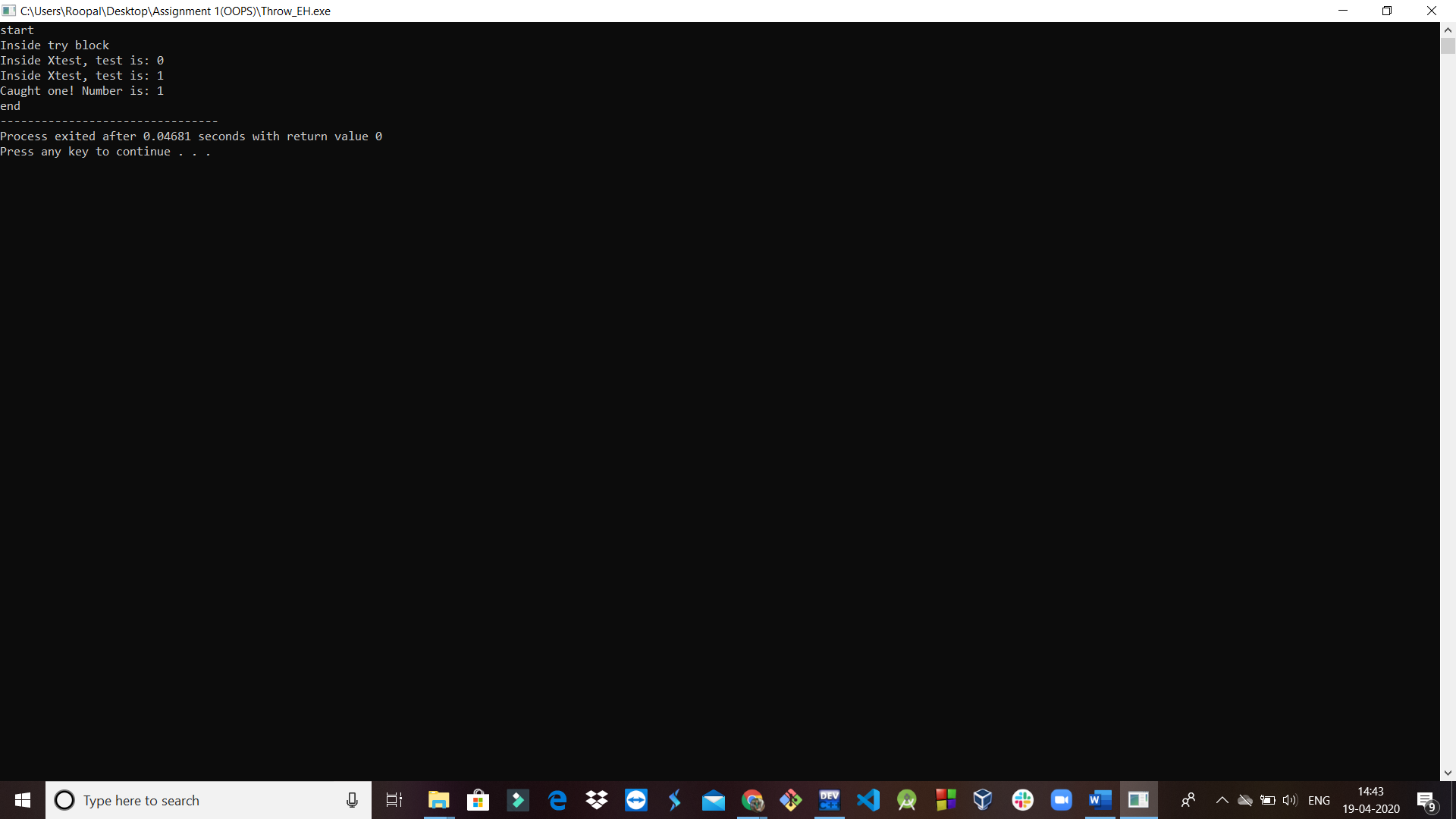
cout << i << "\n";

}

cout << "end";

return 0;

}



**4(d). Rethrow** (using throw keyword)

#include<iostream>

using namespace std;

void divide(double x, double y)

{

cout<<"Inside function \n";

try

{

if(y==0.0)

throw y; //throwing double

else

cout<<"Division = "<<x/y<<"\n";

}

catch(double)

{

cout<<"Caught double inside function \n";

throw; //rethrowing double

}

cout<<"End of function\n\n";

}

int main()

{

cout<<"Inside main \n";

try

{

divide(10.5, 2.0);

divide(20.0, 0.0);

}

catch(double)

{

cout<<"Caught double inside main \n";

}

cout<<"End of main \n";

return 0;

}



**4(e). New**

#include <iostream> // std::cout

#include <new> // std::nothrow

int main ()

{

std::cout << "Attempting to allocate 1 MiB... ";

char\* p = new (std::nothrow) char [1048576];

if (!p) {

// null pointers are implicitly converted to false

std::cout << "Failed!\n";

}

else {

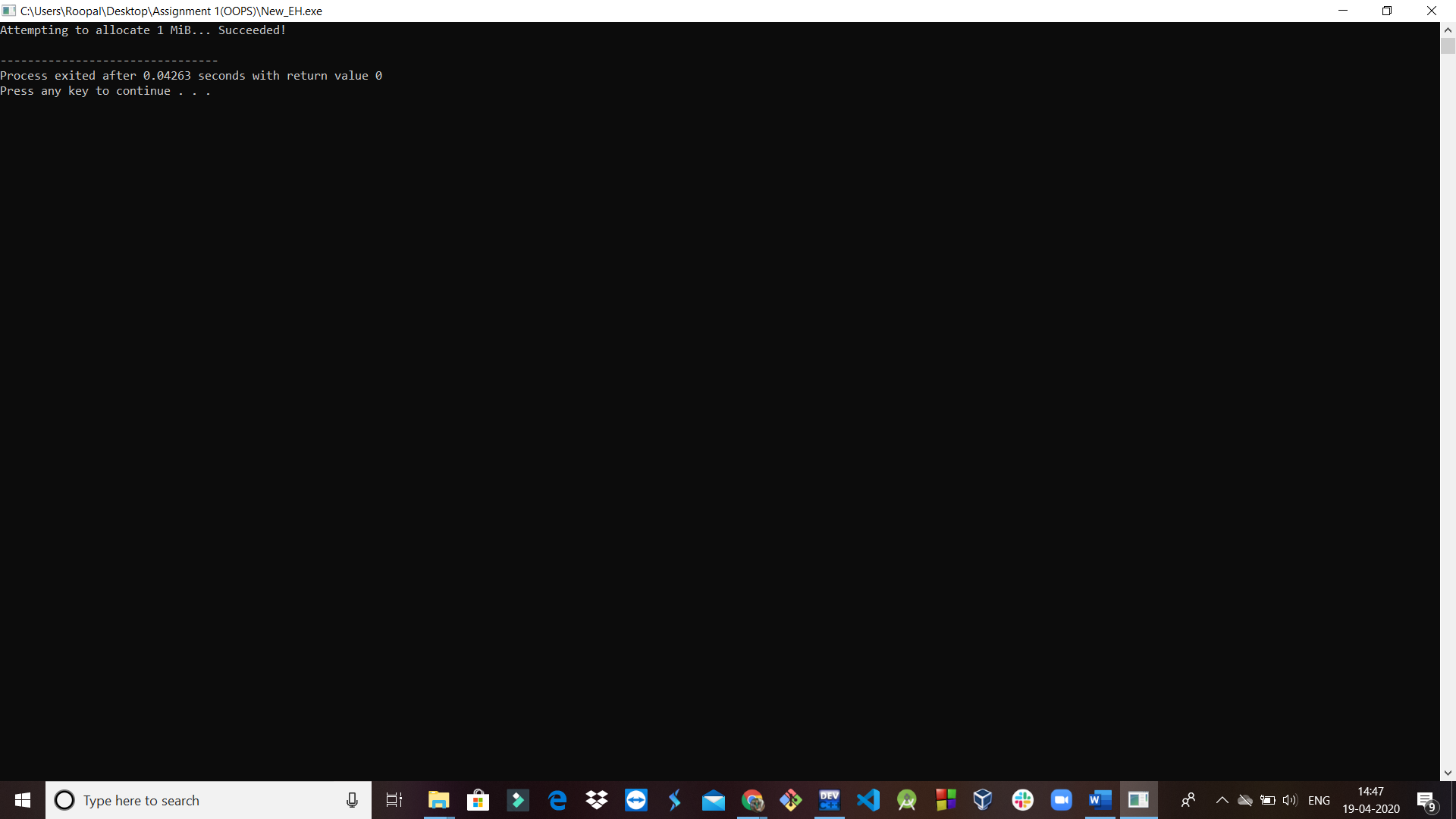
std::cout << "Succeeded!\n";

delete[] p;

}

return 0;

}



**5(a) File IO: open and close a file**

#include<fstream>

#include<iostream>

using namespace std;

int main()

{

ofstream fout;

ifstream fin;

char fname[20];

cout<<"Enter file name: ";

cin.get(fname,20);

fout.open(fname, ios::out);

if(!fout)

{

cout<<"Error in opening the file";

}

else

{

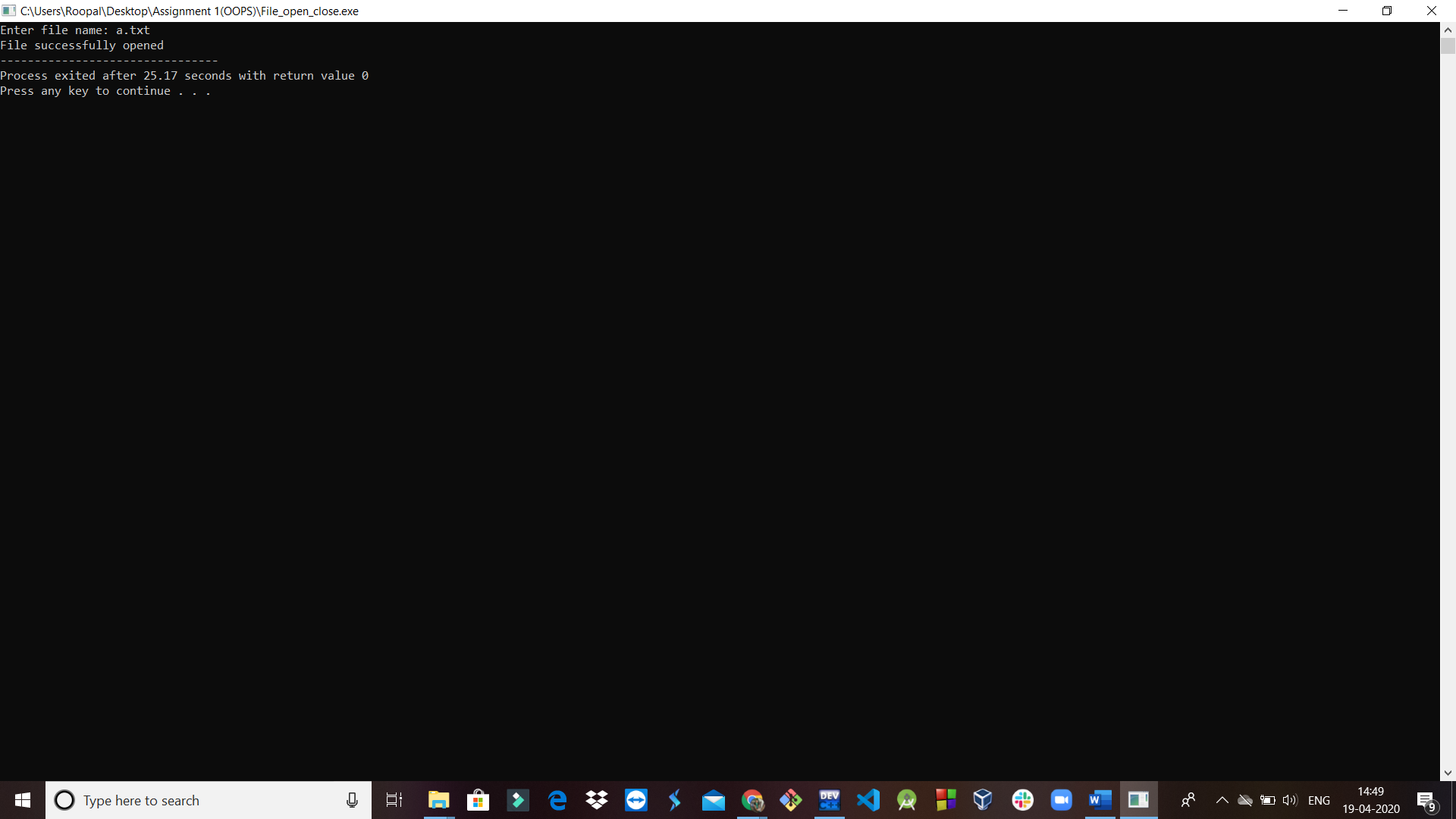
cout<<"File successfully opened";

}

fout.close();

return 0;

}



**5(b) Working with Single File**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ofstream outf("ITEM.txt");

cout<<"Enter item name: ";

char name[30];

cin>>name;

outf<<name<<"\n";

cout<<"Enter item cost: ";

float cost;

cin>>cost;

outf<<cost<<"\n";

ifstream inf("ITEM.txt");

inf>>name;

inf>>cost;

cout<<"\n";

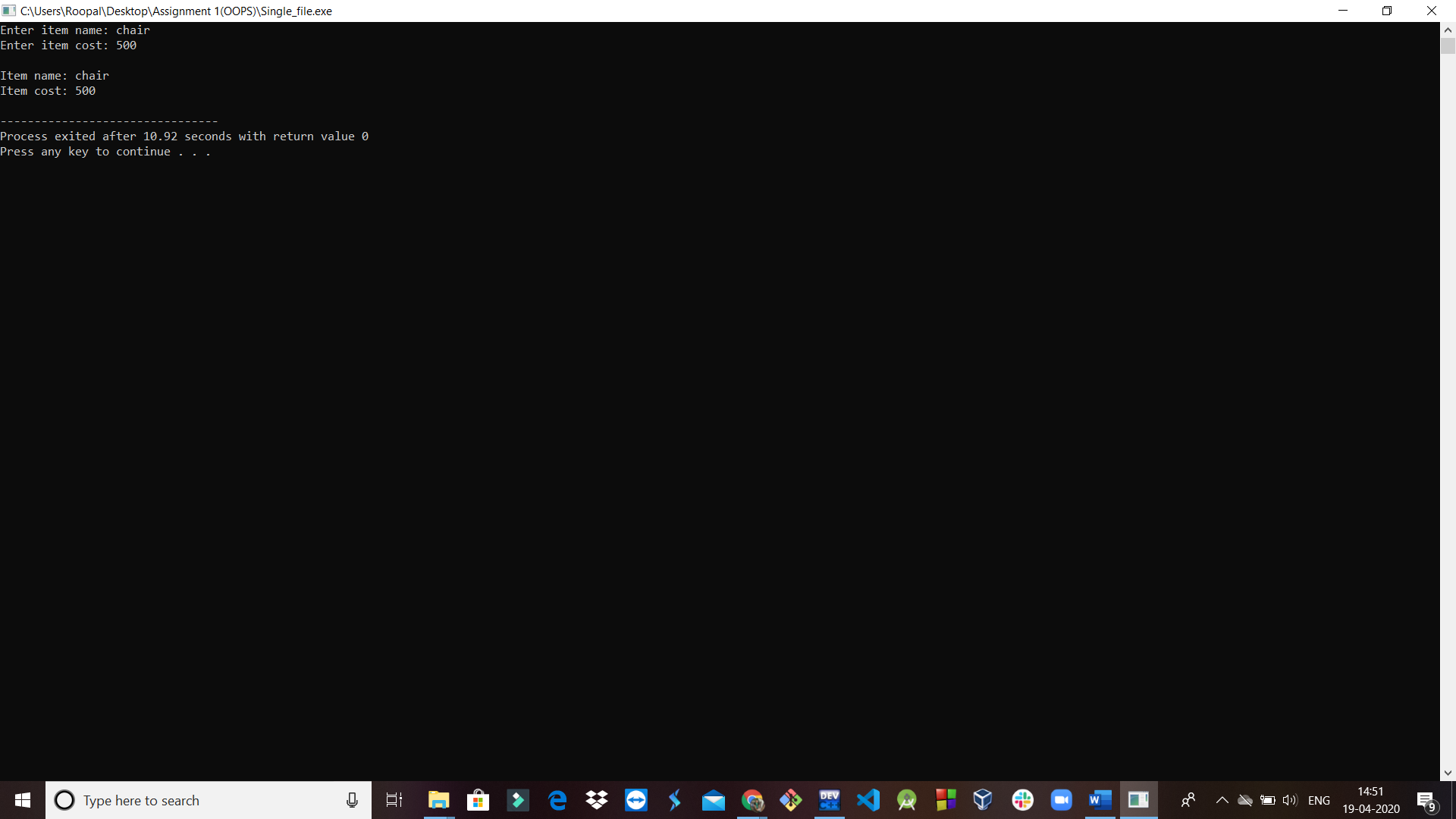
cout<<"Item name: "<<name<<"\n";

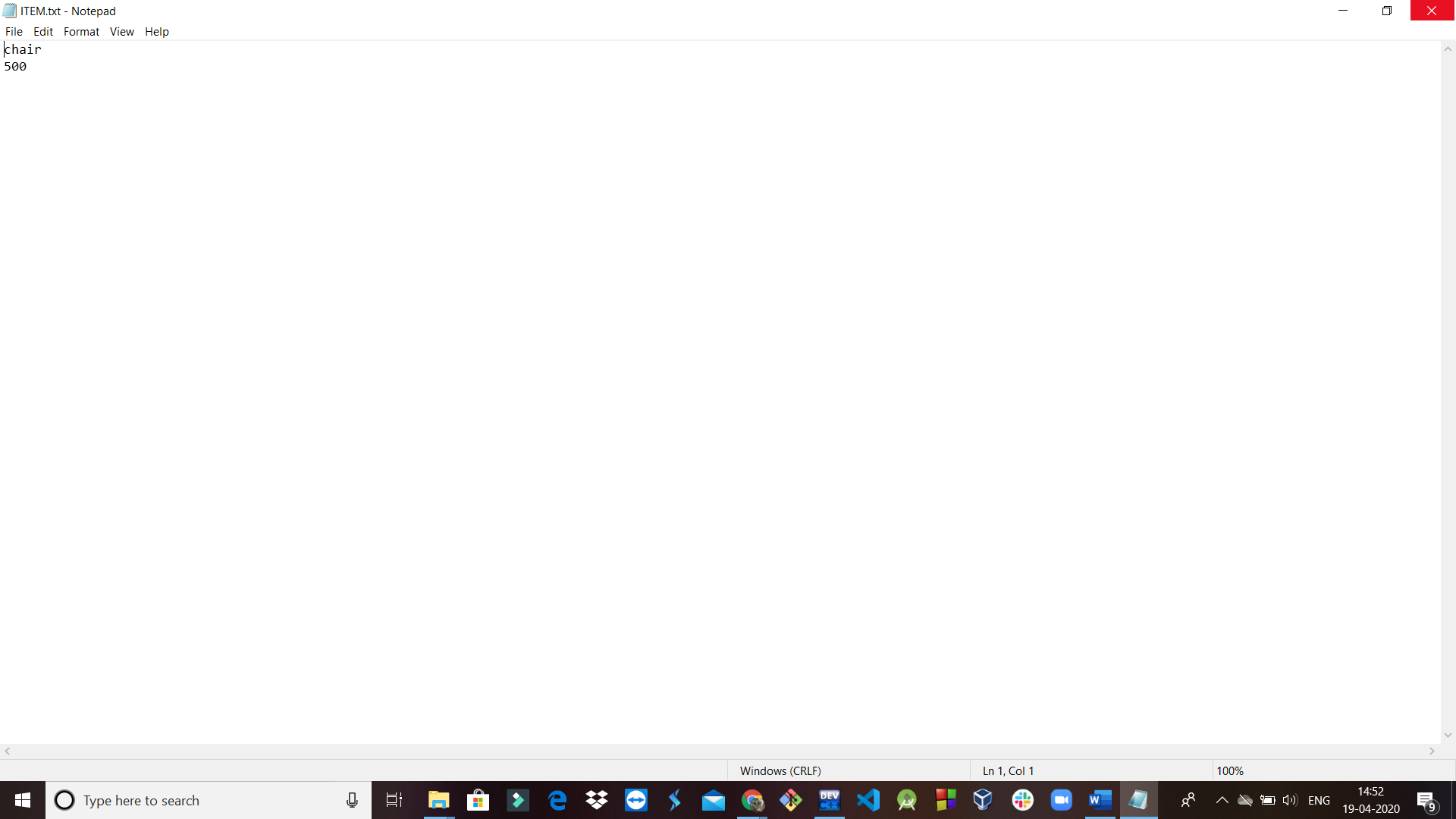
cout<<"Item cost: "<<cost<<"\n";

inf.close();

return 0;

}





**5(c) Working with Multiple Files**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

ofstream fout; //create output stream

fout.open("country.txt"); //connect to country

fout<<"United States of America \n";

fout<<"United Kingdom \n";

fout<<"South Korea \n";

fout.close(); //disconnect country

fout.open("capital.txt"); //connect country

fout<<"Washington DC \n";

fout<<"London \n";

fout<<"Seoul \n";

fout.close(); //disconnect capital

const int N=80; //size of line

char line[N];

ifstream fin; //create input stream

fin.open("country.txt"); //connect country to it

cout<<"Contents of country file\n";

while(fin) //checking end of file

{

fin.getline(line, N); //read a line

cout<<line; //display it

}

fin.close(); //disconnect country

fin.open("capital.txt"); //connect country to it

cout<<"Contents of capital file\n";

while(fin) //checking end of file

{

fin.getline(line, N); //read a line

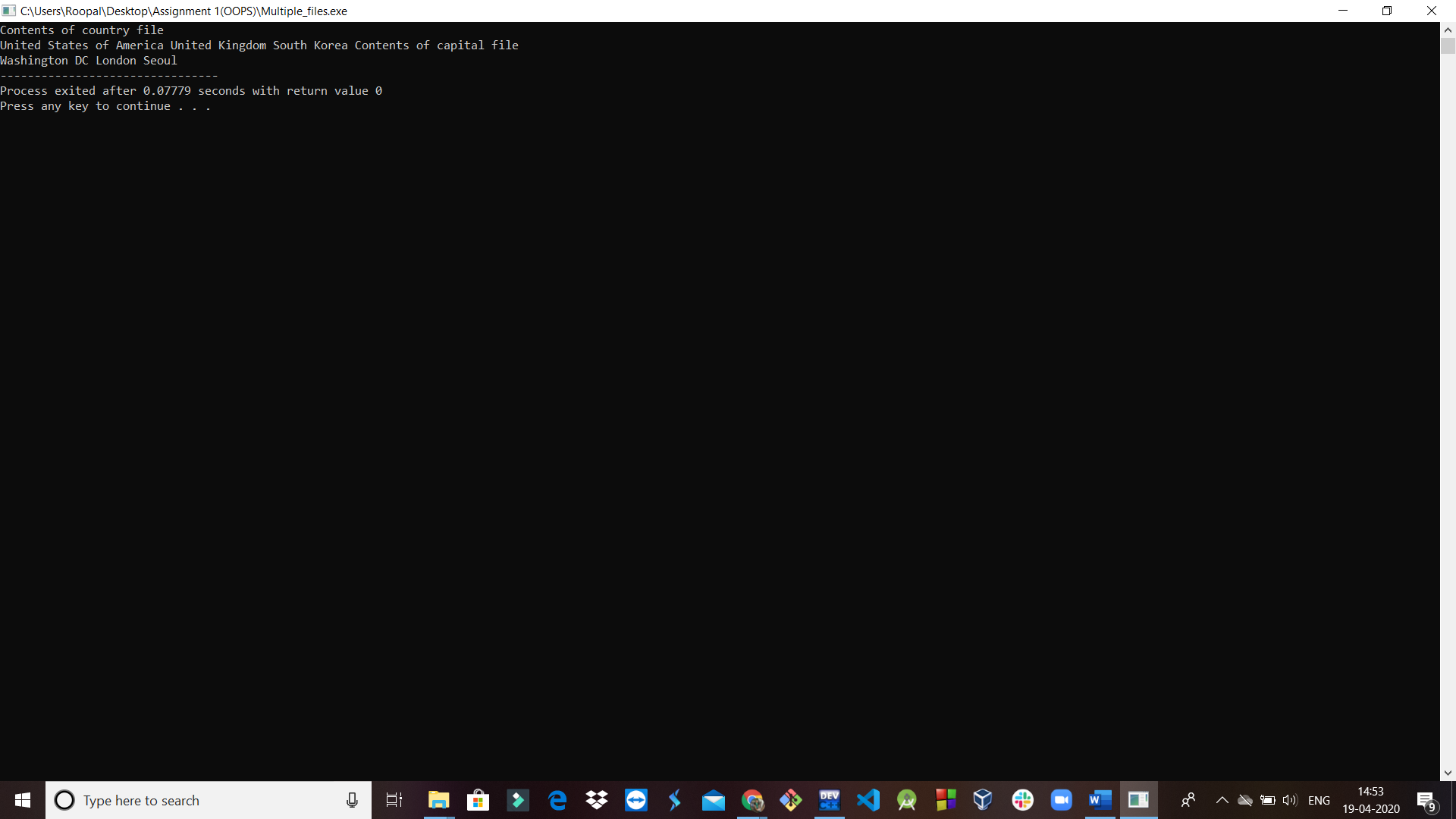
cout<<line; //display it

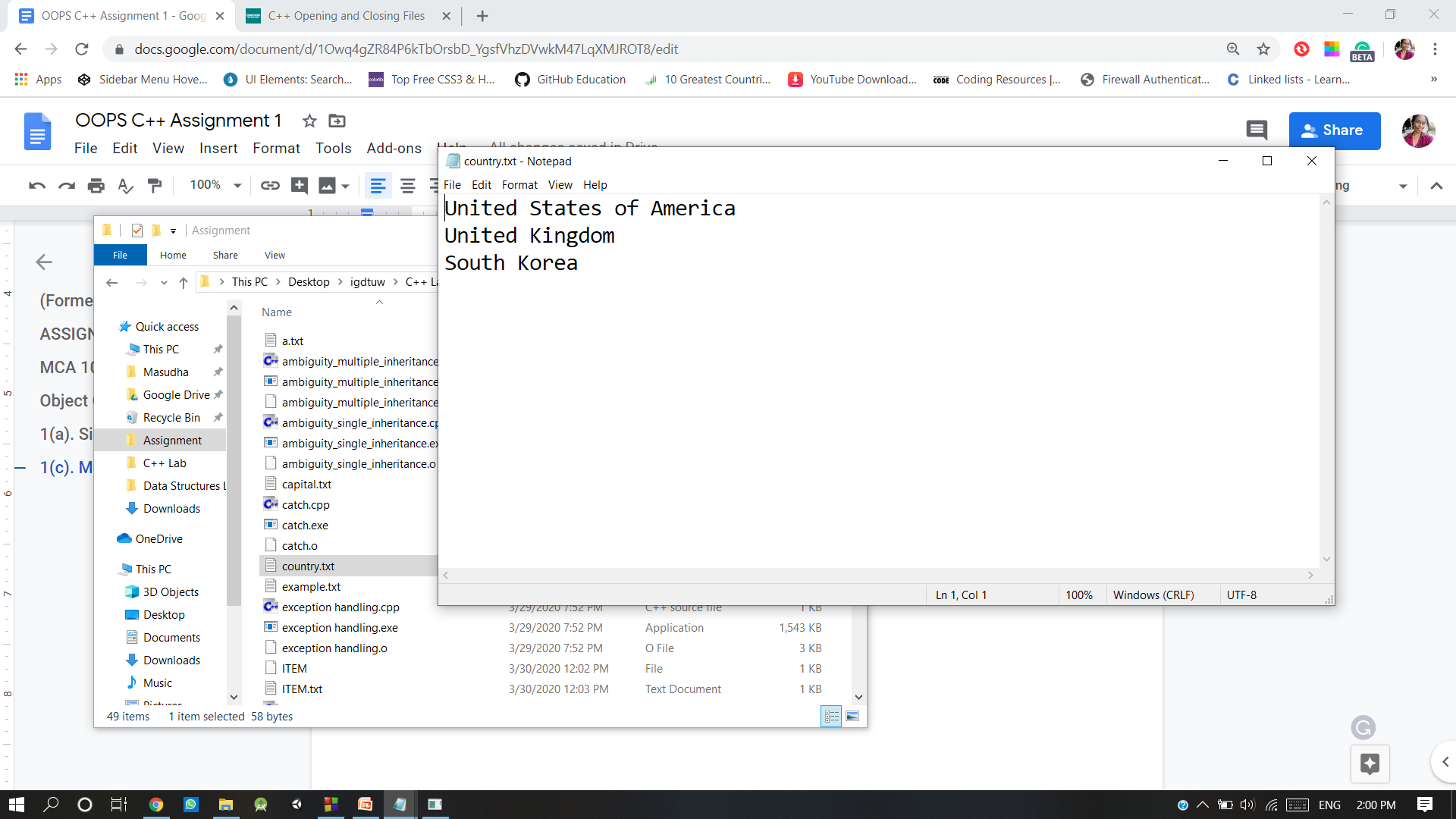
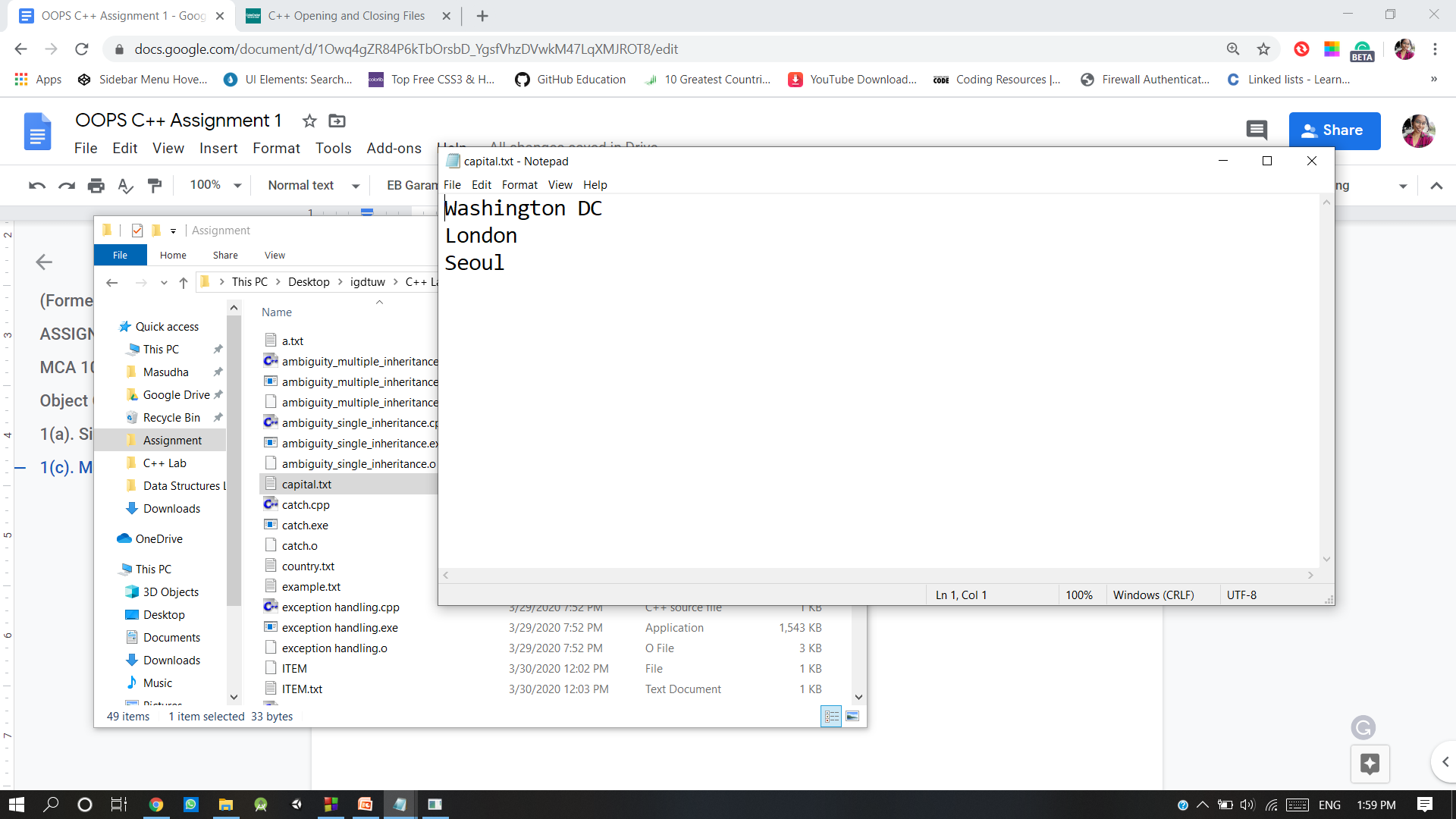
}

fin.close();

return 0;

}



****

**5(d) File size**

#include<iostream>

#include<fstream>

using namespace std;

const char\*filename="example.txt";

int main()

{

long l,m;

ifstream file(filename, ios::in| ios::binary);

l=file.tellg();

file.seekg(0, ios::end);

m=file.tellg();

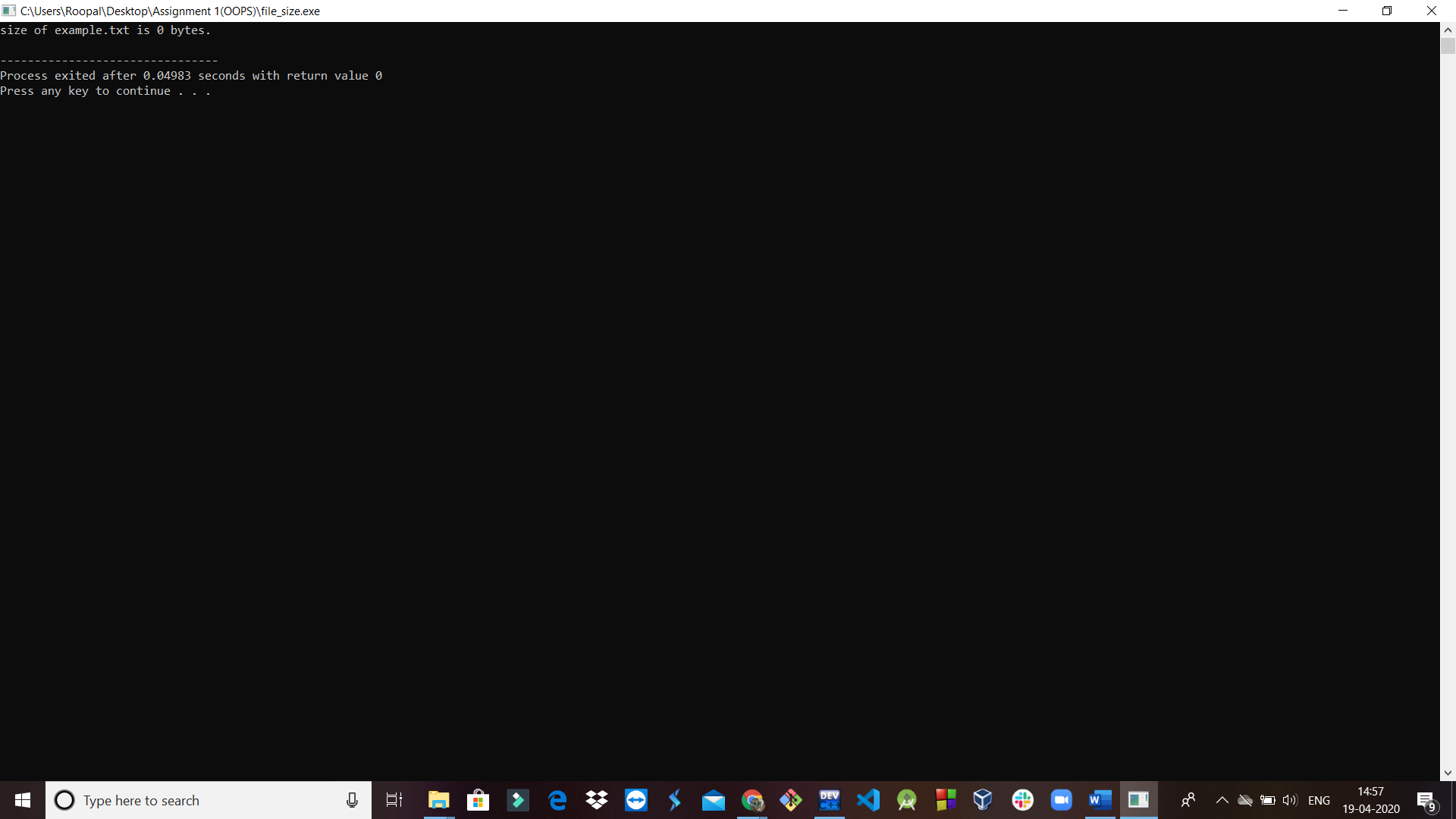
file.close();

cout<<"size of "<<filename;

cout<<" is "<<(m-l)<<" bytes.\n";

return 0;

}



**5(e) Binary File- Read Operation**

#include <iostream>

#include <fstream>

using namespace std;

int main ()

{

streampos size;

char \* memblock;

ifstream file ("example.bin",ios::in|ios::binary|ios::ate);

if (file.is\_open())

{

size = file.tellg();

memblock = new char [size];

file.seekg (0, ios::beg);

file.read (memblock, size);

file.close();

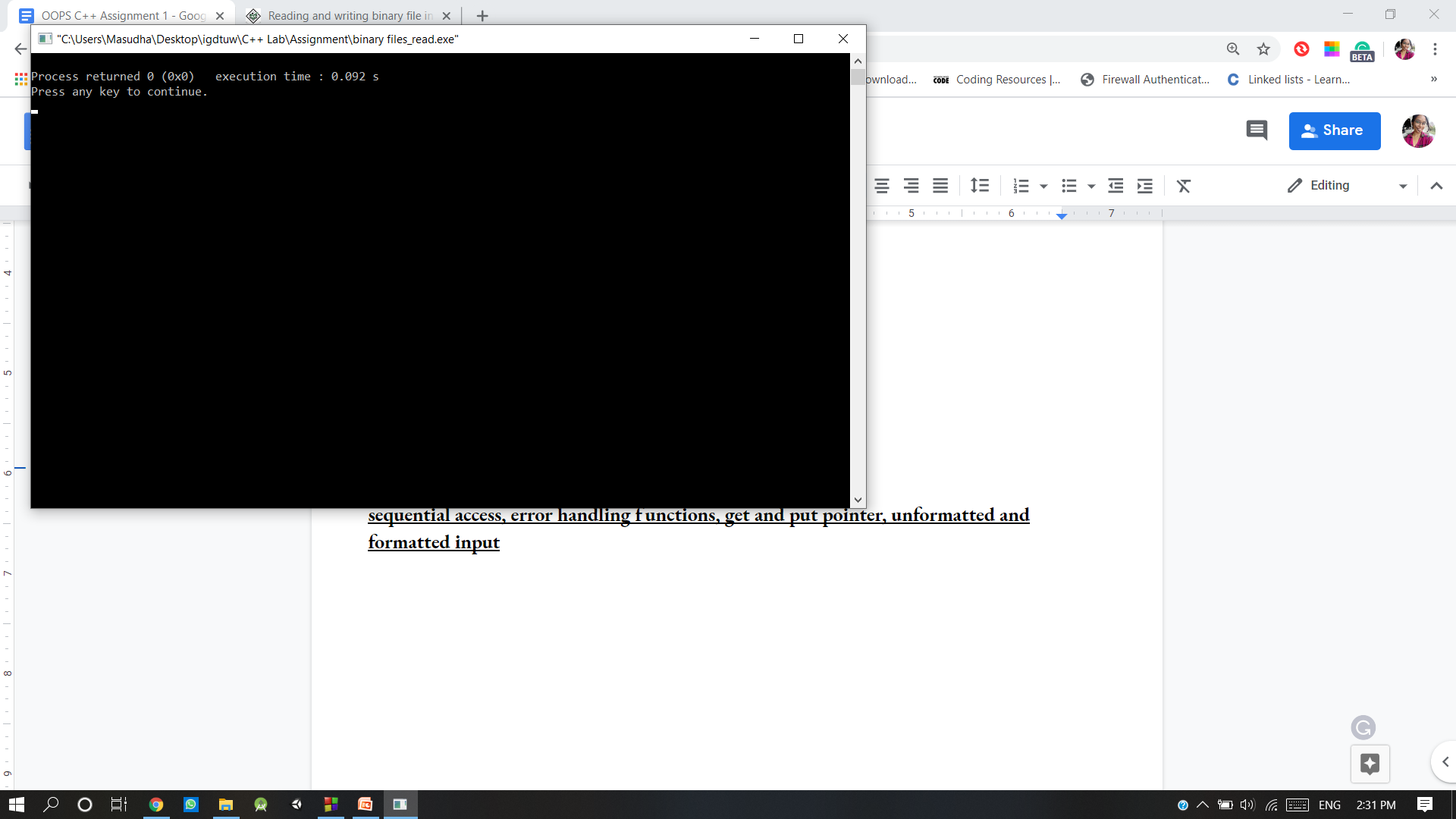
}

else

cout << "Unable to open file";

return 0;

}

****

**5(f) Binary File- Write Operation**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

int arr[3];

int marks[]={98,99,100};

fstream file;

file.open("temp.txt",ios::out|ios::binary);

file.write((char \*)marks, sizeof(marks));

file.close();

file.open("temp.txt",ios::out|ios::binary);

file.read((char \*)arr, sizeof(marks));//read data

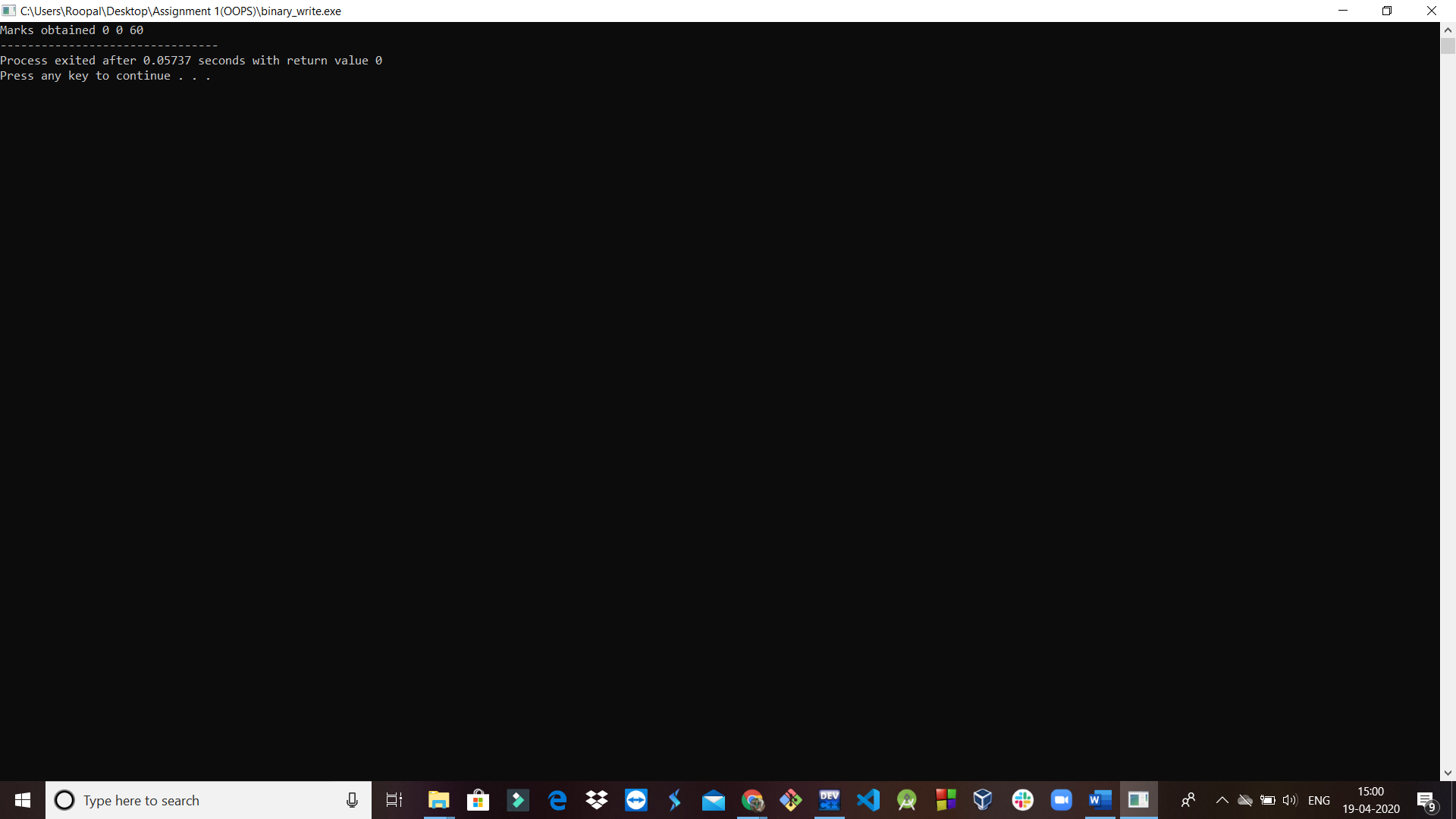
cout<<"Marks obtained ";

for(int i=0;i<3;i++)

cout<<arr[i]<<" ";

file.close();

}



**5(g)Input and Output using get and put**

#include<iostream>

using namespace std;

int main()

{

int count=0;

char c;

cout<<"INPUT TEXT\n";

cin.get(c);

while(c!='\n')

{

cout.put(c);

count++;

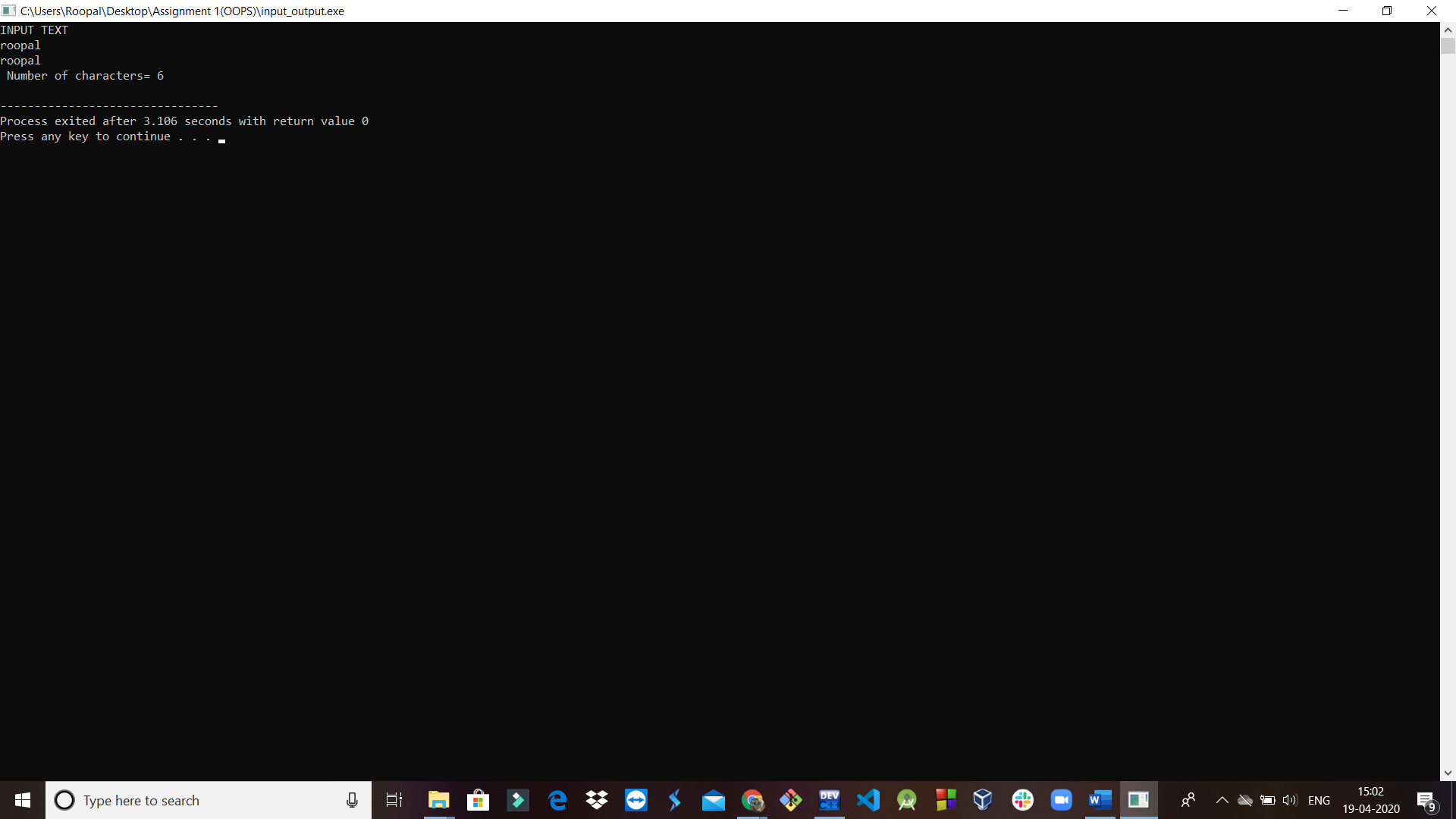
cin.get(c);

}

cout<<"\n Number of characters= "<<count<<"\n";

return 0;

}



**5(h)Getline()**

#include<iostream>

using namespace std;

int main()

{

int size=20;

char city[20];

cout<<"Enter city name:\n";

cin>>city;

cout<<"City name: "<<city<<"\n\n";

cout<<"Enter city name again:\n";

cin.getline(city,size);

cout<<"City name now: "<<city<<"\n\n";

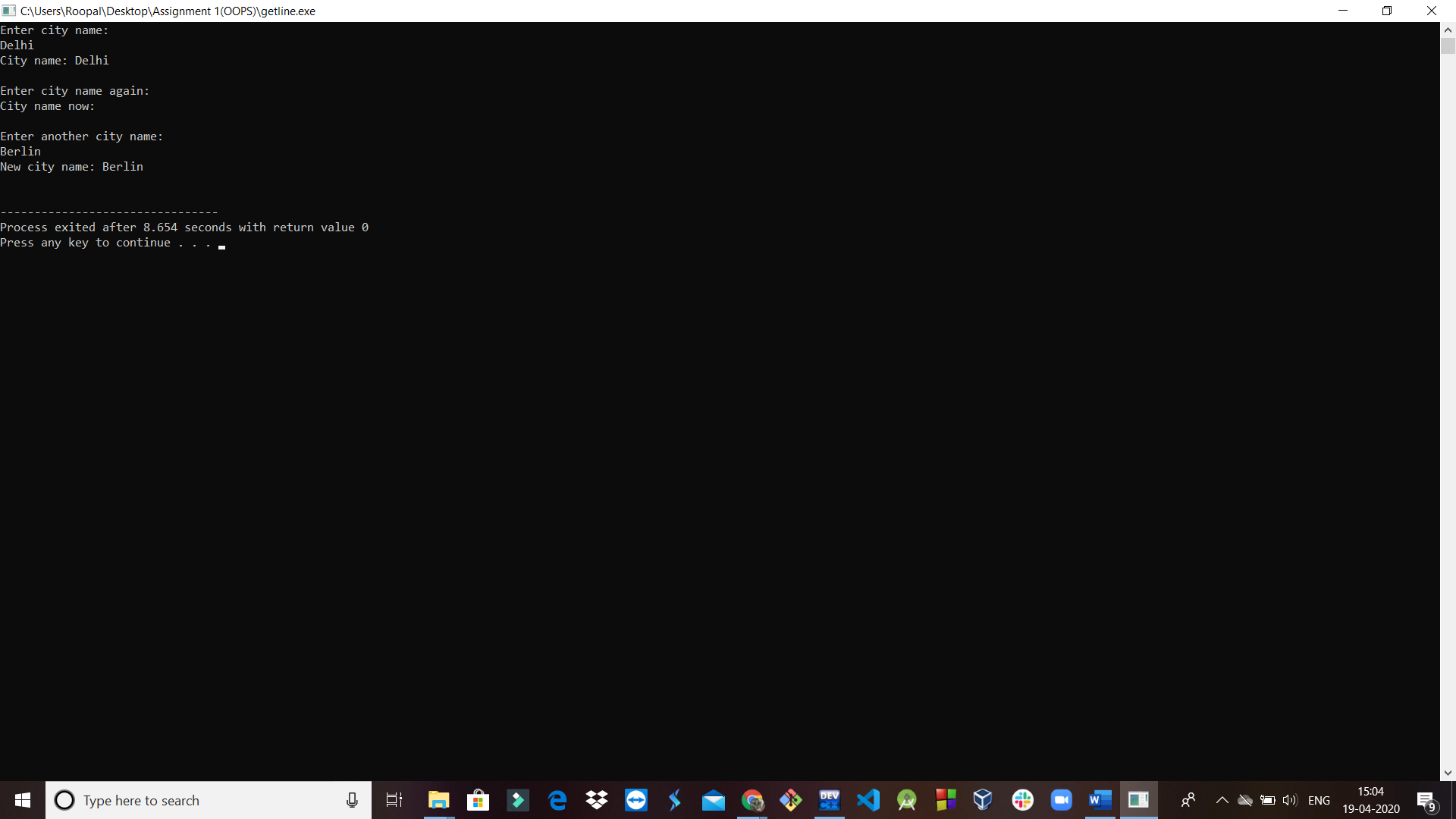
cout<<"Enter another city name:\n";

cin.getline(city,size);

cout<<"New city name: "<<city<<"\n\n";

return 0;

}



**5(i) Unformatted and formatted input**

**Unformatted input:**

**get()**

#include<iostream>

using namespace std;

int main()

{

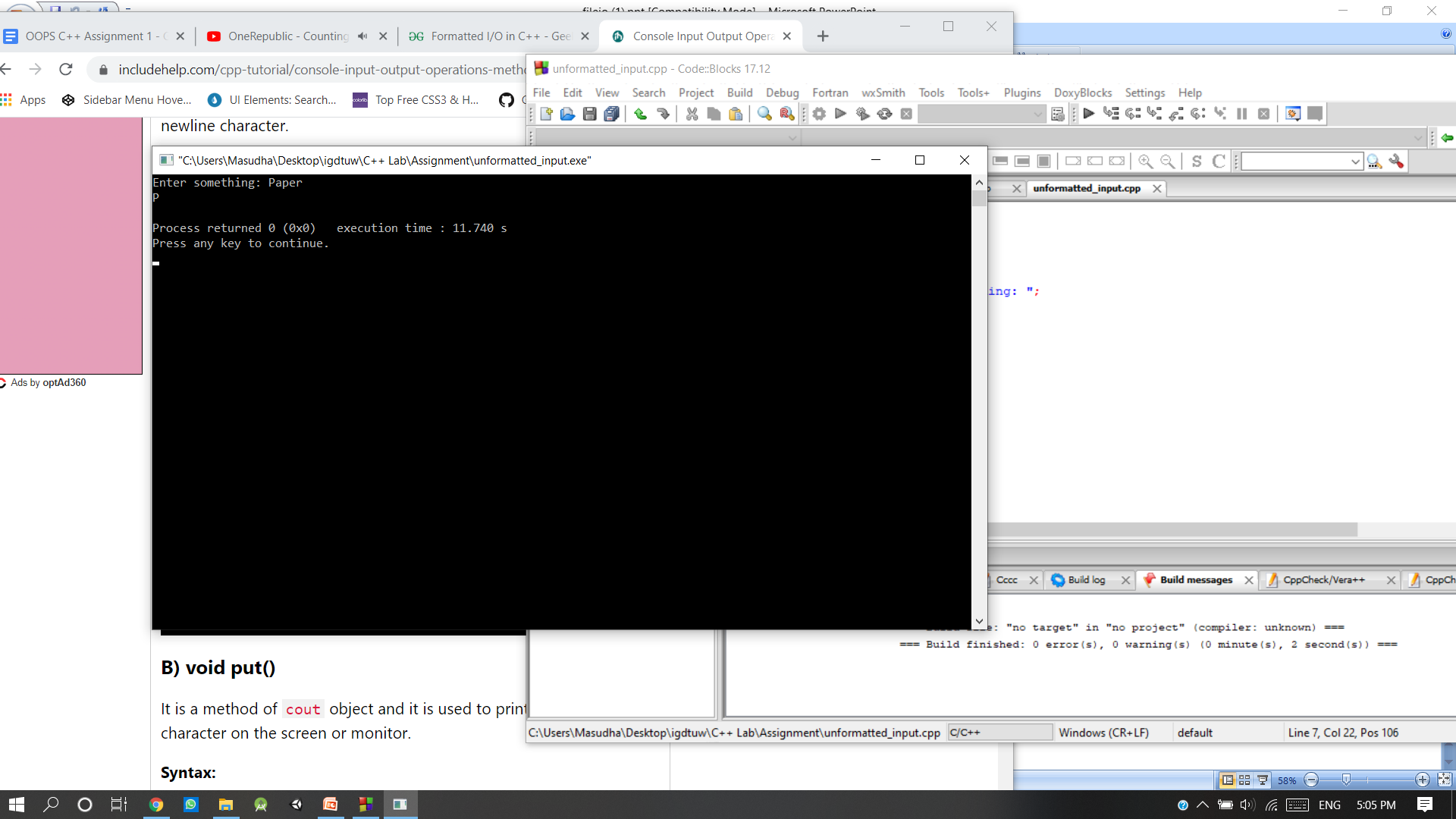
cout<<"Enter something: ";

char c=cin.get();

cout<<c<<endl;

return 0;

}

****

**put()**

#include<iostream>

using namespace std;

int main()

{

cout<<"Enter something: ";

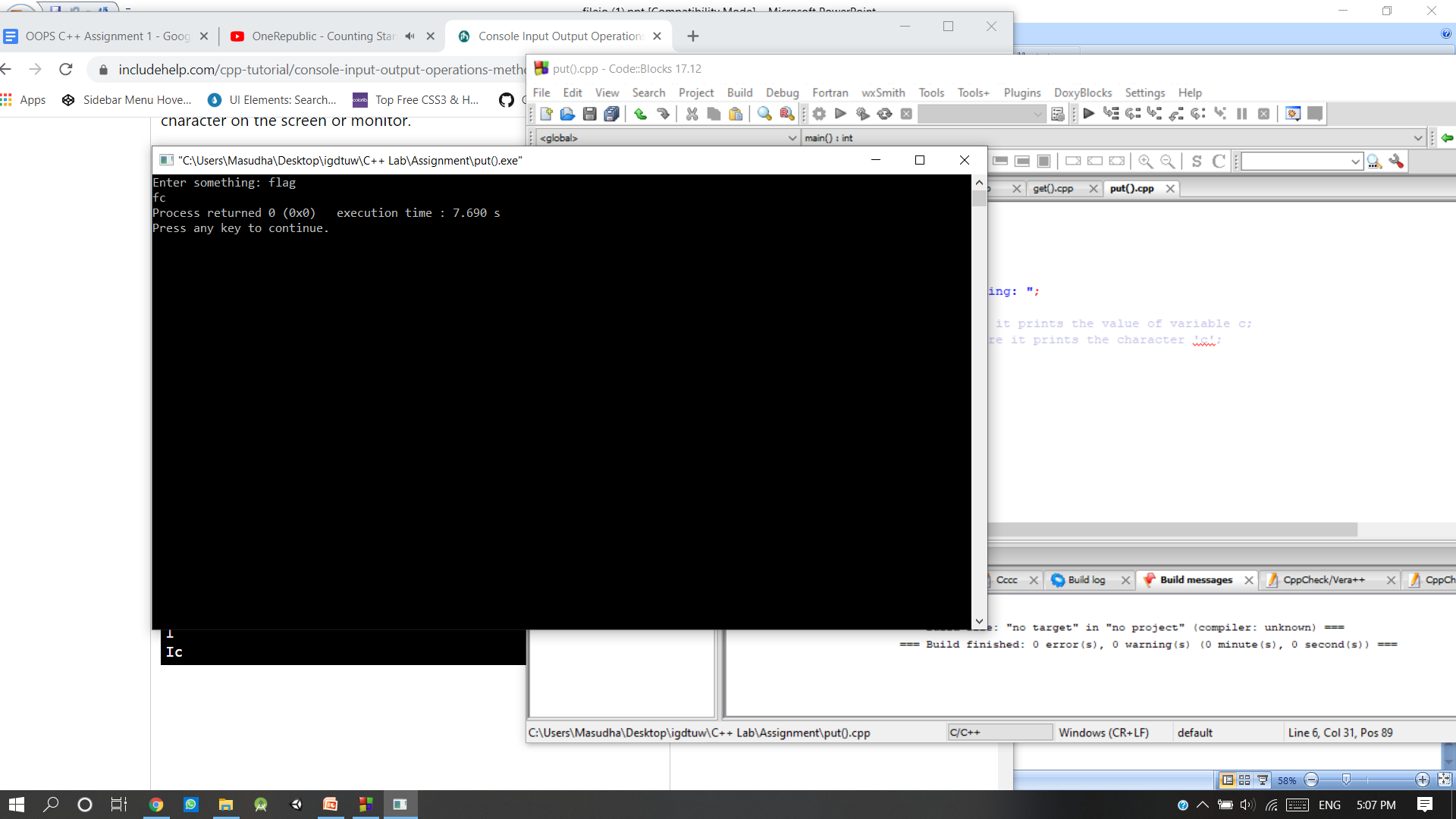
char c=cin.get();

cout.put(c); //Here it prints the value of variable c;

cout.put('c'); //Here it prints the character 'c';

return 0;

}

****

**getline()**

#include<iostream>

using namespace std;

int main()

{

cout<<"Enter name :";

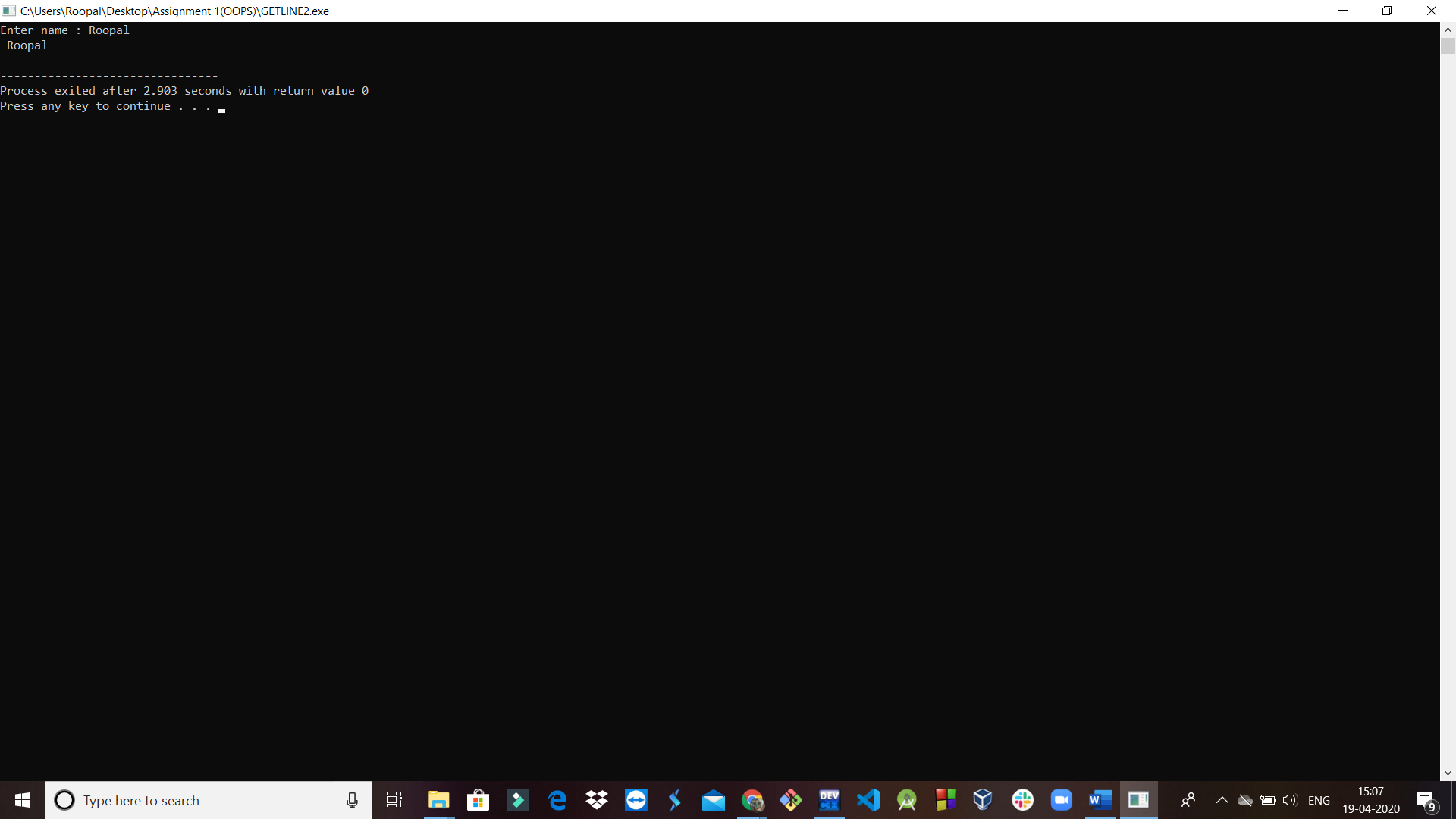
char c[10];

cin.getline(c,10); //It takes 10 characters as input;

cout<<c<<endl;

return 0;

}



**write()**

#include<iostream>

using namespace std;

int main()

{

cout<<"Enter name : ";

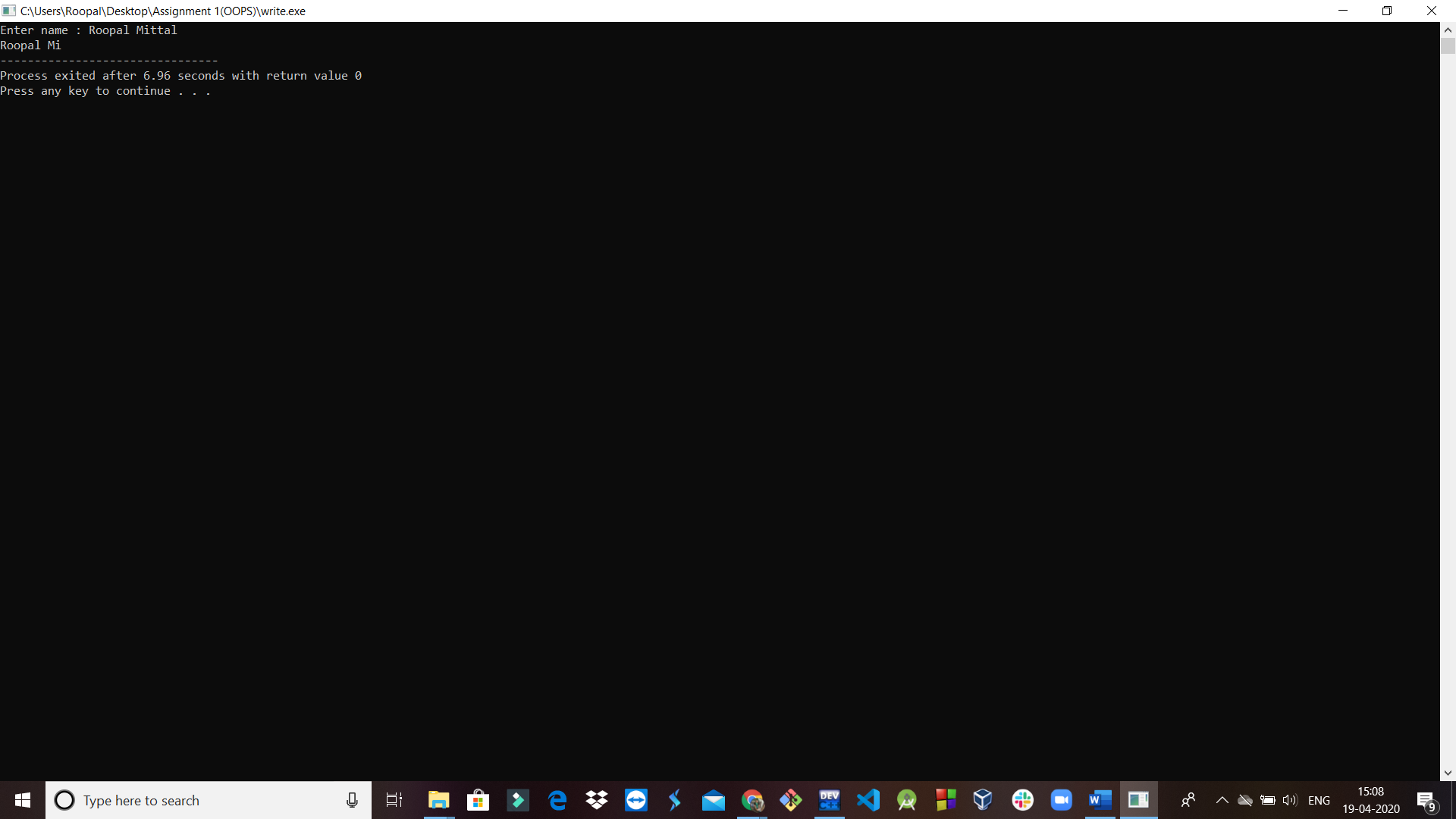
char c[10];

cin.getline(c,10); //It takes 10 charcters as input;

cout.write(c,9); //It reads only 9 character from buffer c;

return 0;

}



**Formatted Input:**

**width()**

#include<iostream>

#include<iomanip>

using namespace std;

int main()

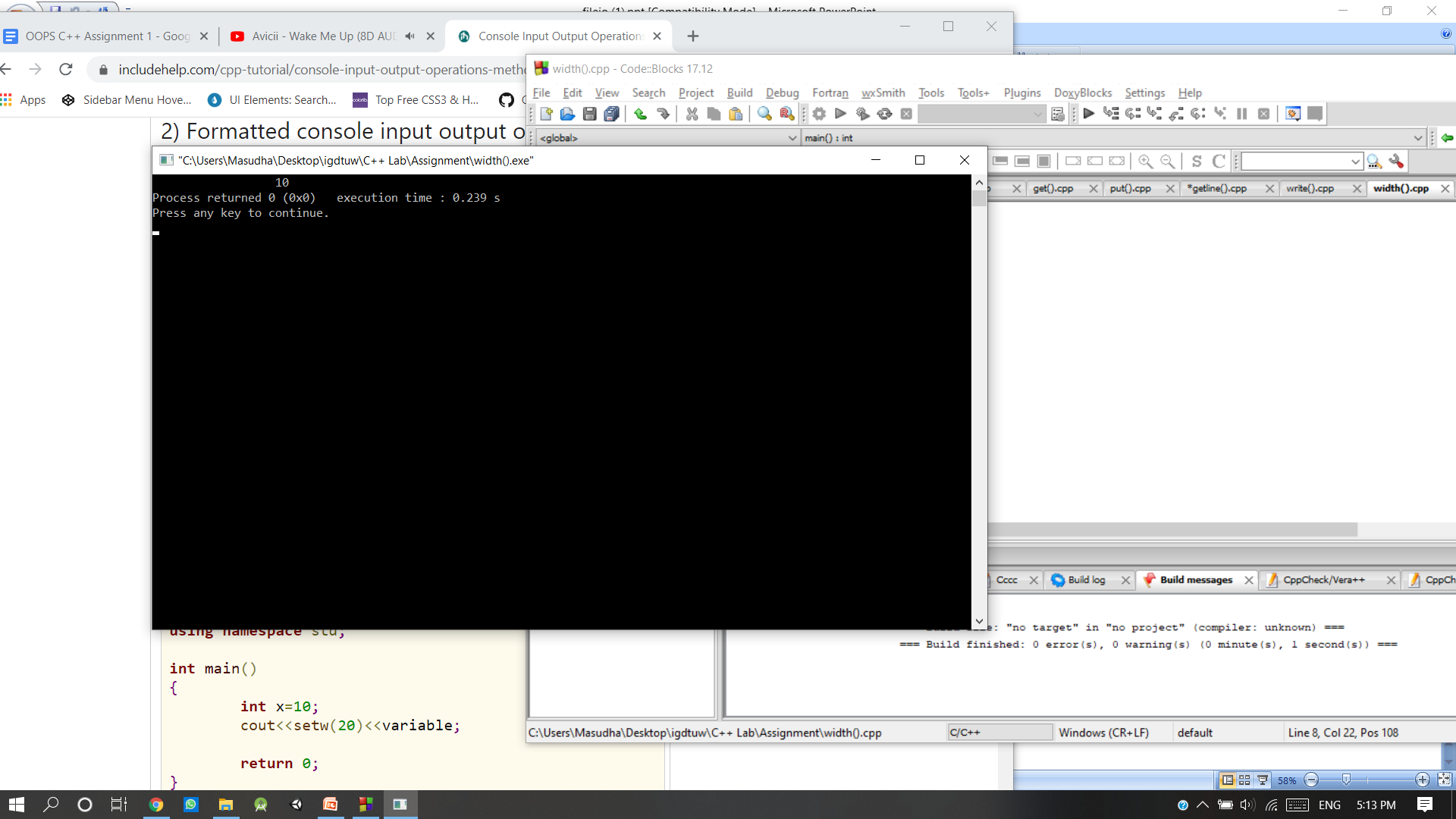
{

int x=10;

cout<<setw(20)<<x;

return 0;

}



**fill()**

#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

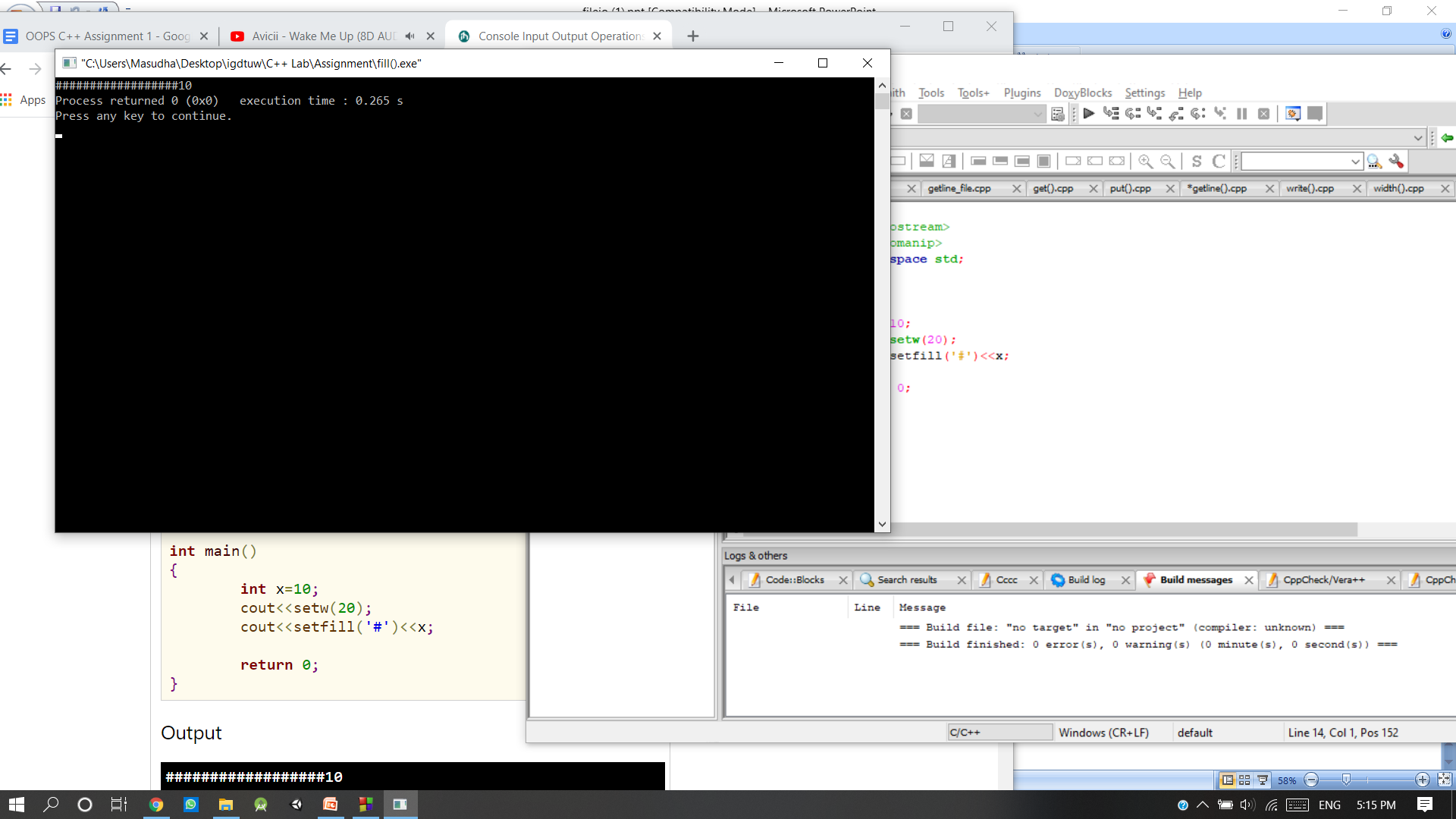
int x=10;

cout<<setw(20);

cout<<setfill('#')<<x;

return 0;

}



**precision()**

#include<iostream>

#include<iomanip>

using namespace std;

int main()

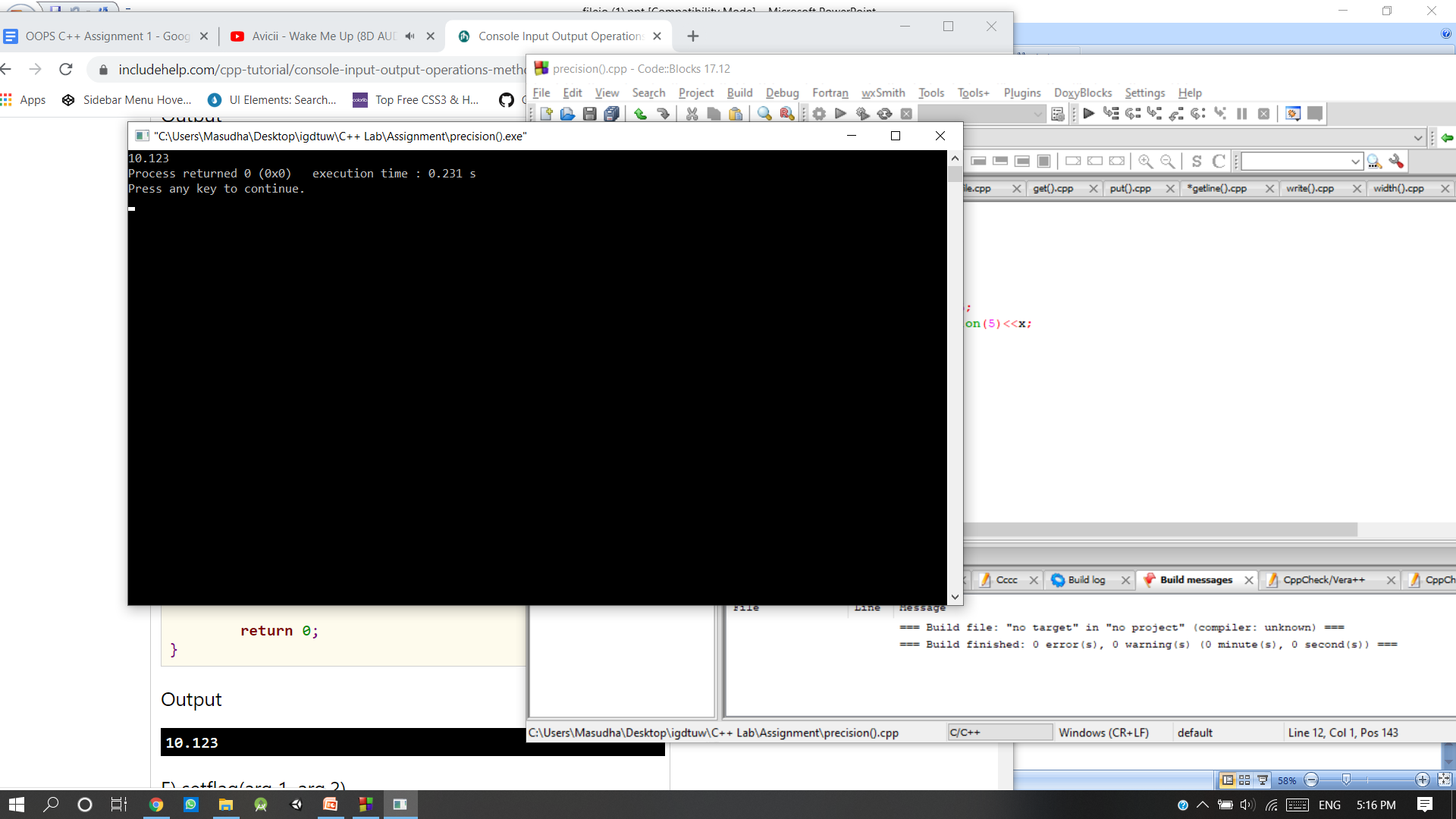
{

float x=10.12345;

cout<<setprecision(5)<<x;

return 0;

}

****