OBJECT ORIENTED PROGRAMMING LAB

ETCS-258

**Faculty name**: Ms. Akansha Kochhar

**Student name:** ARIHANT JAIN

**Roll No.:**08914802719

**Group:** 4C-5



Maharaja Agrasen Institute of Technology, PSP area,

Sector – 22, Rohini, New Delhi – 110085

INDEX OF THE CONTENTS

|  |  |  |
| --- | --- | --- |
| S.No. | EXPERIMENTS | REMARKS |
| **1.** | To write a program for multiplication of two matrices using Object-Oriented Programming (OOP). |  |
| **2.** | To write a program to perform addition of two complex numbers using constructor overloading. The first constructor which takes no argument is used to create objects which are not initialized, second which takes one argument is used to initialize real and image parts to equal values and third which takes two argument is used to initialized real and image to two different values. |  |
| **3.** | To write a C++ program to find the greater of two given private numbers in two different classes using a friend function. |  |
| **4.** | To implement a class string containing the following functions:   1. Overload + operator to carry out the concatenation of strings. 2. Overload = operator to carry out string copy. 3. Overload <= operator to carry out the comparison of strings. 4. Function to display the length of string. 5. Function tolower() to convert upper case to lower case. 6. Function toupper() to convert lower case letters to upper case. |  |
| **5.** | To create a class called LIST with two pure virtual functions store () and retrieve (). To store a value call store and to retrieve, call retrieve function. Derive two classes stack and queue from it and override store and retrieve. |  |
| **6.** | To write a program to define the function template for calculating the square of given numbers with different data types. |  |
| **7.** | To write a program to demonstrate the use of special functions, constructor and destructor in the class template. The program is used to find the bigger of two entered numbers. |  |
| **8.** | To write a program to perform the deletion of white spaces such as horizontal tab, vertical tab, space, line feed, new line and carriage return from a text file and store the contents of the file without the white spaces on another file. |  |
| **9.** | To write a program to read the class object of student info such as name, age, sex, height and weight from the keyboard and to store them on a specified file using read( ) and write( ) functions. Again, the same file is opened for reading and displaying the contents of the file on the screen. |  |
| **10.** | To write a program to raise an exception if any attempt is made to refer to an element whose index is beyond the array size. |  |
| **11.** | To write a C++ program to find whether a number is prime or not. |  |
| **12.** | To write a program to take name, address as character string, age as int, salary as float and contains an inline function to set the values and display it. |  |
| **13.** | Using the concept of function overloading, to write a function for calculating area of triangle, circle and rectangle. |  |
| **14.** | To create a class Student, which have data members as name, branch, roll no., age, sex, five subjects. Display the name of the student & his percentage who has more than 70%. |  |
| **15.** | To create a class TIME with members hours, minutes and seconds. Take input, add two-time objects and passing objects to function and display the result. |  |
| **16.** | To create a function power to raise a number m to a power n. The function takes a double value for m and integer value for n. Use the default value for n to make the function. Calculate the squares when this argument is omitted. |  |
| **17.** | To write a program to enter any number and find its factorial using a constructor. |  |
| **18.** | To write a program to generate a Fibonacci series using Copy Constructor. |  |
| **19.** | To write a program to find the biggest of three numbers using a Friend Function. |  |
| **20.** | To write a program to demonstrate the use of a friend function with Inline assignment. |  |

Experiment-1

**AIM:** To write a program for multiplication of two matrices using Object-Oriented Programming (OOP).

**THEORY:** Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

Languages that support object-oriented programming (OOP) typically use inheritance for code reuse and extensibility in the form of either classes or prototypes. Those that use classes support two main concepts:

* Classes – the definitions for the data format and available procedures for a given type or class of object; may also contain data and procedures (known as class methods) themselves, i.e. classes contain the data members and member functions
* Objects – instances of classes

**SYNTAX:**

// Class definition

class <class-name>

{

<access-specifier>:

<data-members>

};

// Object creation

<class-name> <object-name>;

**Source Code:**

#include <iostream>

using namespace std;

/\*

Matrix Class with private members

ROWS, COLS, and MATRIX itself

Functions:

set\_rows() [setter]

set\_cols() [setter]

input() [inputs matrix]

multiply() [multiplies two matrices]

print() [prints matrix]

\*/

class Matrix

{

int ROWS, COLS;

int MATRIX[100][100];

public:

// Default constructor

Matrix()

{

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

for (int j = 0; j < COLS; j++)

MATRIX[i][j] = 0;

}

// Sets rows

void set\_rows(int R)

{

ROWS = R;

}

// Sets columns

void set\_cols(int C)

{

COLS = C;

}

// Inputs matrix

void input()

{

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

{

cout << "Row " << i+1 << ": ";

for (int j = 0; j < COLS; j++)

{

cin >> MATRIX[i][j];

}

}

}

// Multiplies two matrices and return the result

Matrix multiply(Matrix M)

{

if (COLS != M.COLS)

{

cout << "Matrix multiplication not possible.\n";

return \*this;

}

Matrix ANS;

ANS.set\_rows(ROWS);

ANS.set\_cols(M.COLS);

// Multiply respective rows and columns and store

// answer in ANS matrix

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

for (int j = 0; j < M.COLS; j++)

for (int k = 0; k < COLS; k++)

ANS.MATRIX[i][j] = ANS.MATRIX[i][j] + (MATRIX[i][k] \* M.MATRIX[k][j]);

return (Matrix) ANS;

}

// Prints matrix

void print()

{

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

{

for (int j = 0; j < COLS; j++)

{

cout << MATRIX[i][j] << "\t";

}

cout << "\n";

}

}

};

int main()

{

Matrix A, B;

int r, c;

// Input rows and columns

cout << "Enter the number of rows of Matrix A: ";

cin >> r;

A.set\_rows(r);

cout << "Enter the number of columns of Matrix A: ";

cin >> c;

A.set\_cols(c);

// Input rows and columns

cout << "Enter the number of rows of Matrix B: ";

cin >> r;

B.set\_rows(r);

cout << "Enter the number of columns of Matrix B: ";

cin >> c;

B.set\_cols(c);

// Input matrix

cout << "\n| Enter Matrix A |\n\n";

A.input();

// Input matrix

cout << "\n| Enter Matrix B |\n\n";

B.input();

// Create new object and store result in it

Matrix C;

C = A.multiply(B);

// Print multiplied matrix

cout << "\n| Multiplied matrix |\n\n";

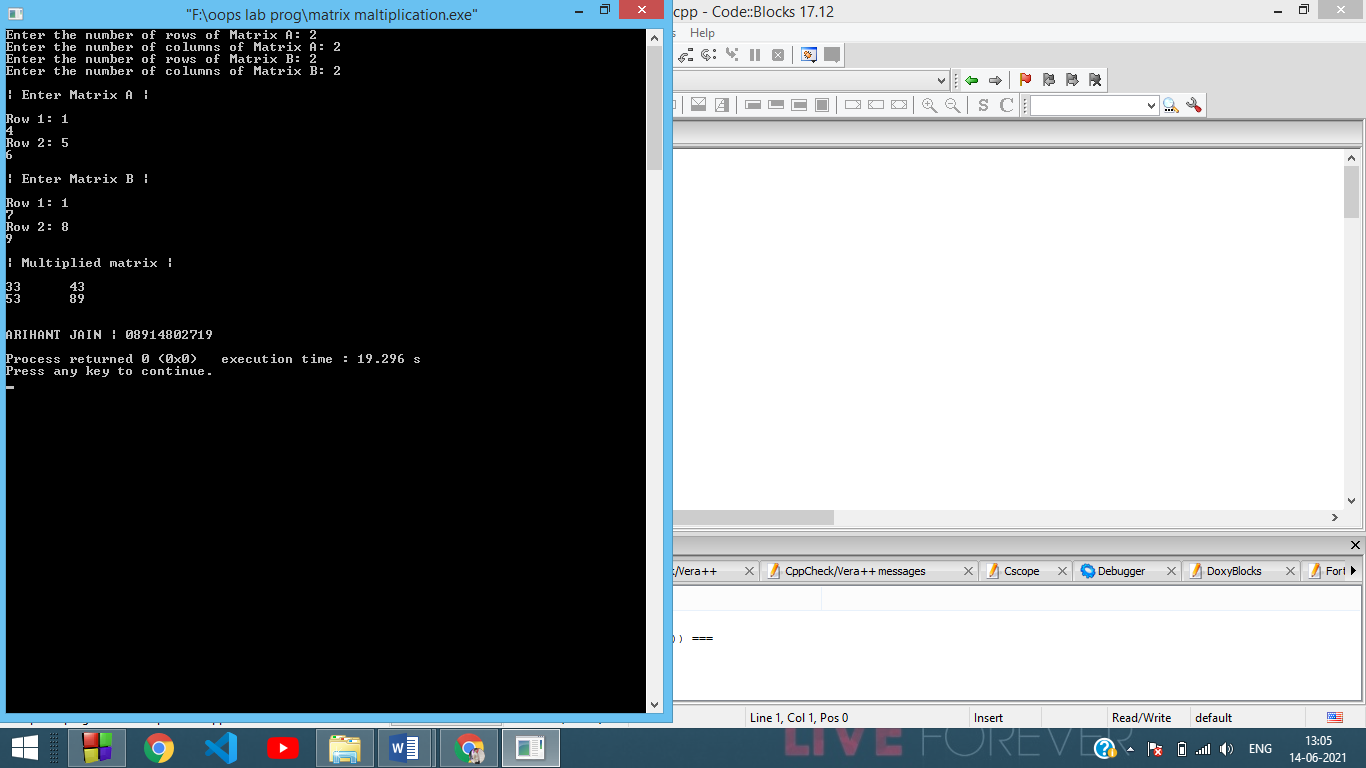
C.print();

cout << "\n\nARIHANT JAIN | 08914802719\n";

return 0;

}

**OUTPUT:**

****

Experiment-2

**AIM:** To write a program to perform addition of two complex numbers using constructor overloading. The first constructor which takes no argument is used to create objects which are not initialized, second which takes one argument is used to initialize real and imag parts to equal values and third which takes two argument is used to initialized real and imag to two different values.

**THEORY**

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class. Constructor has same name as the class itself, don’t have return type, and are automatically called when an object is created.

Types of constructors:

* **Default Constructor:** It is the constructor which doesn’t take any argument. It has no parameters.
* **Parameterised Constructor:** It is the constructor which doesn’t take any argument. It can take any number of arguments.
* **Copy Constructor:** It is a member function which initializes an object using another object of the same class. It copies the object’s data.

**SYNTAX:**

// Default constructor

A() { }

// Parameterised constructor

A(<parameter-1>, <parameter-2>) { }

// Copy constructor

A(A obj) { // Copy obj data to ‘this’ object }

**Source Code:**

#include <iostream>

using namespace std;

// Complex number class

class Complex

{

int real, image;

public:

// Default constructor

Complex()

{

real = 0;

image = 0;

}

// Parameterised constructor

Complex(int val)

{

real = val;

image = val;

}

// Parameterised constructor

Complex(int val1, int val2)

{

real = val1;

image = val2;

}

// Adds two complex numbers and returns result

Complex add(Complex C)

{

real = real + C.real;

image = image + C.image;

return \*this;

}

// Prints complex number as "A + Bi"

void print()

{

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

}

};

int main()

{

// Default constructor used

Complex C;

// Parameterised constructor used

Complex A(-4,6);

// Parameterised constructor used

Complex B(3, -2);

cout << "Complex A: ";

A.print();

cout << "Complex B: ";

B.print();

// Add two complex numbers and store in C

C = A.add(B);

// Print sum

cout << "Complex C (sum): ";

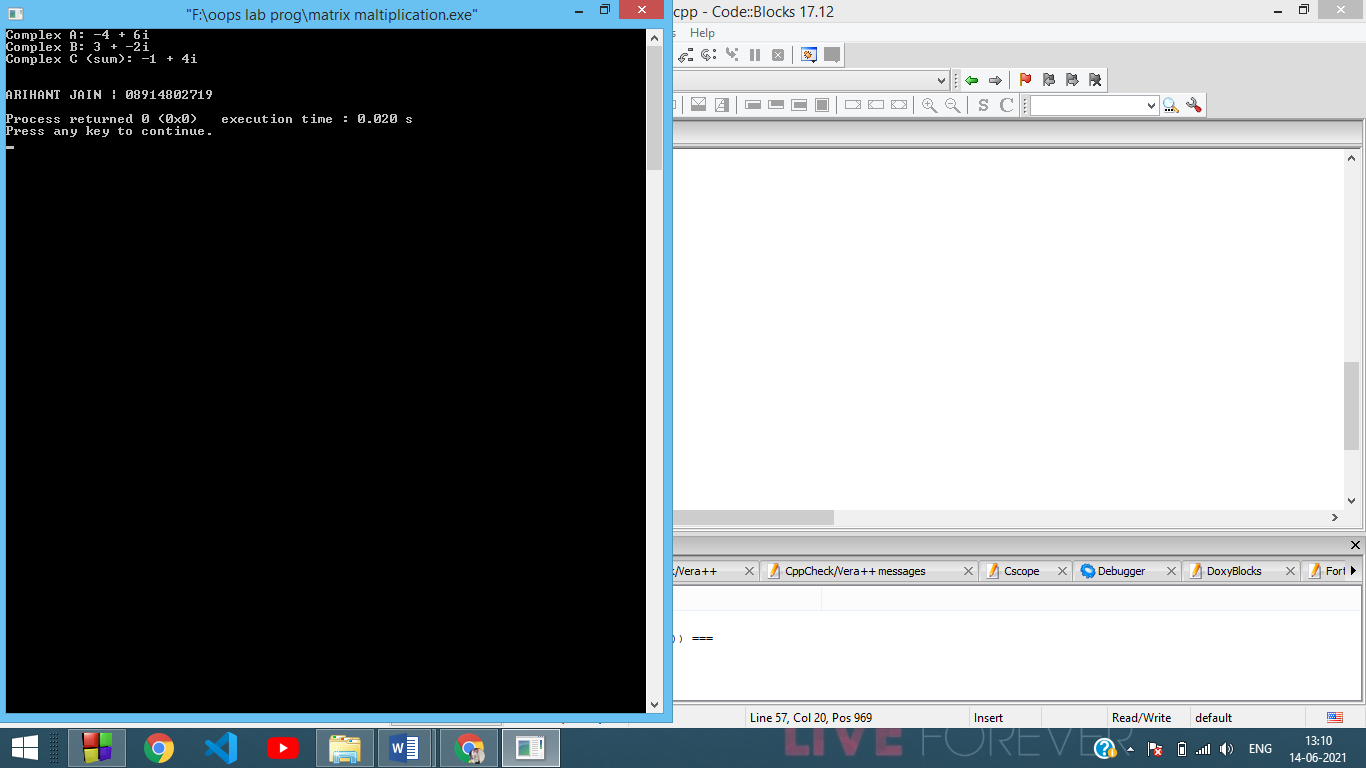
C.print();

cout << "\n\nARIHANT JAIN | 08914802719\n";

return 0;

}

**OUTPUT:**

****

Experiment-3

**AIM:** To write a C++ program to find the greater of two given private numbers in two different classes using a Friend function.

**THEORY:** A Friend Function of a class is a function that has a right to access all private and protected members of a class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions. It is neither mutual, nor is it inherited.

To declare a function as a friend of a class, we precede the function prototype in the class definition with the keyword ‘friend’.

It is sometimes useful to allow a particular class to access private members of other class. Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.

**SYNTAX:**

class A

{

friend <return-type> <func-name>(<parameters>);

}

// Friend function of class A

// Not using friend keyword here

<return-type> <func-name>(<parameters>)

{

// Code on private members of class A

}

**Source Code:**

#include <iostream>

using namespace std;

// Forward declaration of class B

class B;

class A

{

// Private member of A

int num;

public:

// Inputs private member 'num' of A

void input()

{

cout << "Enter num value for A: ";

cin >> num;

}

// Friend function declaration

friend int compare(A, B);

};

class B

{

// Private member of B

int num;

public:

// Inputs private member 'num' of B

void input()

{

cout << "Enter num value for B: ";

cin >> num;

}

friend int compare(A, B);

};

// Friend function

// Classes: A and B

// Returns the private members of both classes which is greater

int compare(A a, B b)

{

return (a.num > b.num ? a.num : b.num);

}

int main()

{

// Create objects of both classes

A a\_obj;

B b\_obj;

// Input objects of both classes

a\_obj.input();

b\_obj.input();

// Output sum by calling friend function

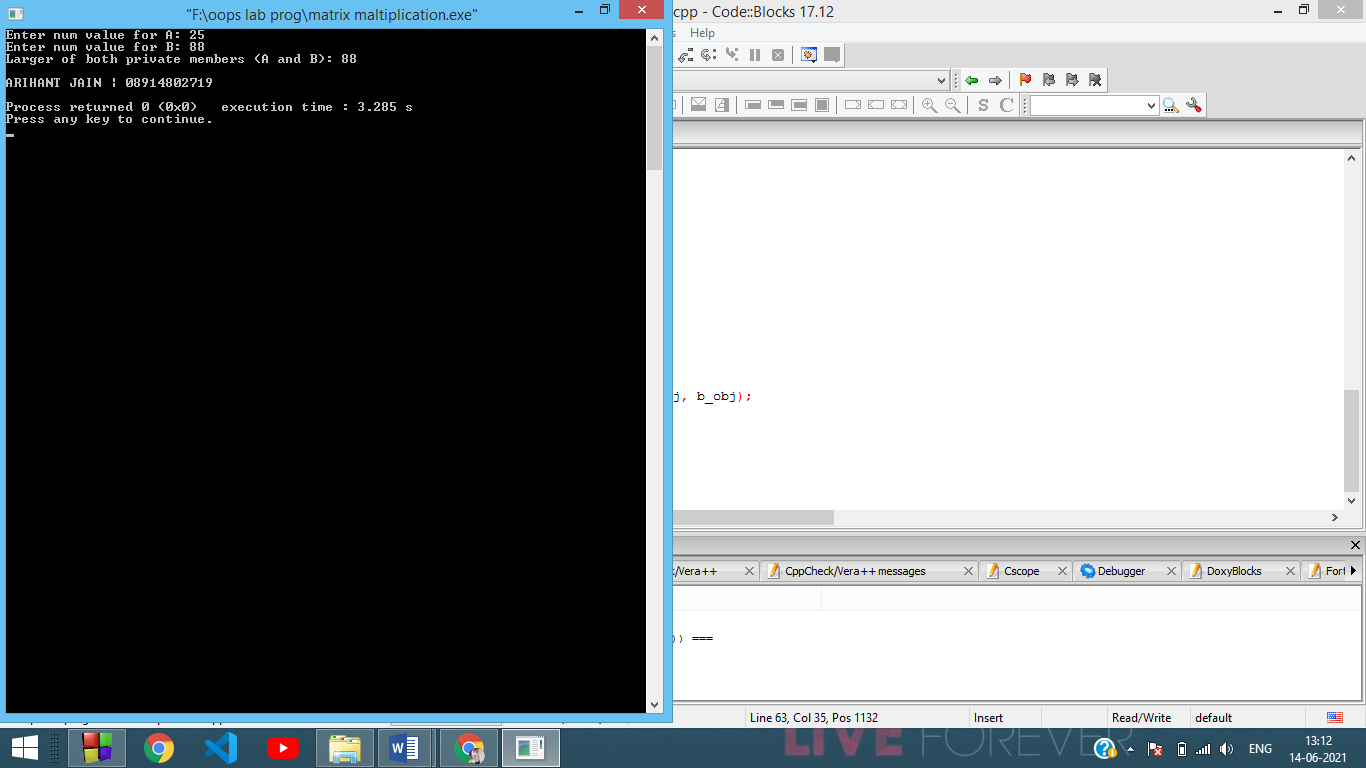
cout << "Larger of both private members (A and B): " << compare(a\_obj, b\_obj);

cout << "\n\nARIHANT JAIN | 08914802719\n";

return 0;

}

**OUTPUT:**

****

Experiment-4

**AIM:** To implement a class string containing the following functions:

1. Overload + operator to carry out the concatenation of strings.
2. Overload = operator to carry out string copy.
3. Overload <= operator to carry out the comparison of strings.
4. Function to display the length of string.
5. Function tolower() to convert upper case to lower case.
6. Function toupper() to convert lower case letters to upper case.

**THEORY**

* **Function overloading** is a feature in C++ where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading “Function” name should be the same and the arguments should be different. Function overloading can be considered as an example of polymorphism feature in C++.
* **Operator Overloading:** In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.  
  Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.

Almost all operators can be overloaded except few. Following is the list of operators that cannot be overloaded:

**. (dot)**

**::**

**?:**

**sizeof**

**SYNTAX:**

class A

{

// Function overloading

function(data-type1 parameter1);

function(data-type2 parameter 1);

function(parameter1, parameter2);

// Operator overloading

return-type operator (+, -, \*, <= etc.) (parameter)

{

// Assign members

}

}

**Source Code:**

#include <iostream>

using namespace std;

#include <cstring>

#include <cctype>

class String

{

public:

// Original string (char array)

char s[100];

// Inputs the original string

String get\_string()

{

std::cin >> s;

return \*this;

}

// Outputs the original string

char\* put\_string()

{

return (char\*) s;

}

// Computes length of string

int length()

{

return strlen(s);

}

// Assigns original strings of one object to another

String operator = (const String &op)

{

strcpy(s, op.s);

return \*this;

}

// Concatenates two strings

String operator + (const String &op)

{

String res;

res = \*this;

strcat(res.s, op.s);

return res;

}

// Compares two strings based on length

bool operator <= (String op)

{

if (length() <= op.length())

return true;

else

return false;

}

// Converts the original string to all lowercase

String tolower()

{

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

{

s[i] = std::tolower(s[i]);

}

return \*this;

}

// Converts the original string to all lowercase

String toupper()

{

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

{

s[i] = std::toupper(s[i]);

}

return \*this;

}

};

int main()

{

cout<<"ARIHANT JAIN(08914802719)"<<endl;

String A, B;

std::cout << "Enter the first string: ";

A.get\_string();

std::cout << "Enter the second string: ";

B.get\_string();

// Assignment and Concatenation

String C = A + B;

std::cout << "\nConcatenated string: " << C.put\_string() << "\n";

// Comparison

std::cout << "String A <= String B: " << (A <= B ? "true\n" : "false\n");

// Length

std::cout << "Length of string A: " << A.length() << "\n";

std::cout << "Length of string B: " << B.length() << "\n";

// tolower and toupper

String A\_lower = A.tolower(), A\_upper = A.toupper();

String B\_lower = B.tolower(), B\_upper = B.toupper();

std::cout << "tolower(A): " << A\_lower.put\_string() << "\n";

std::cout << "toupper(A): " << A\_upper.put\_string() << "\n";

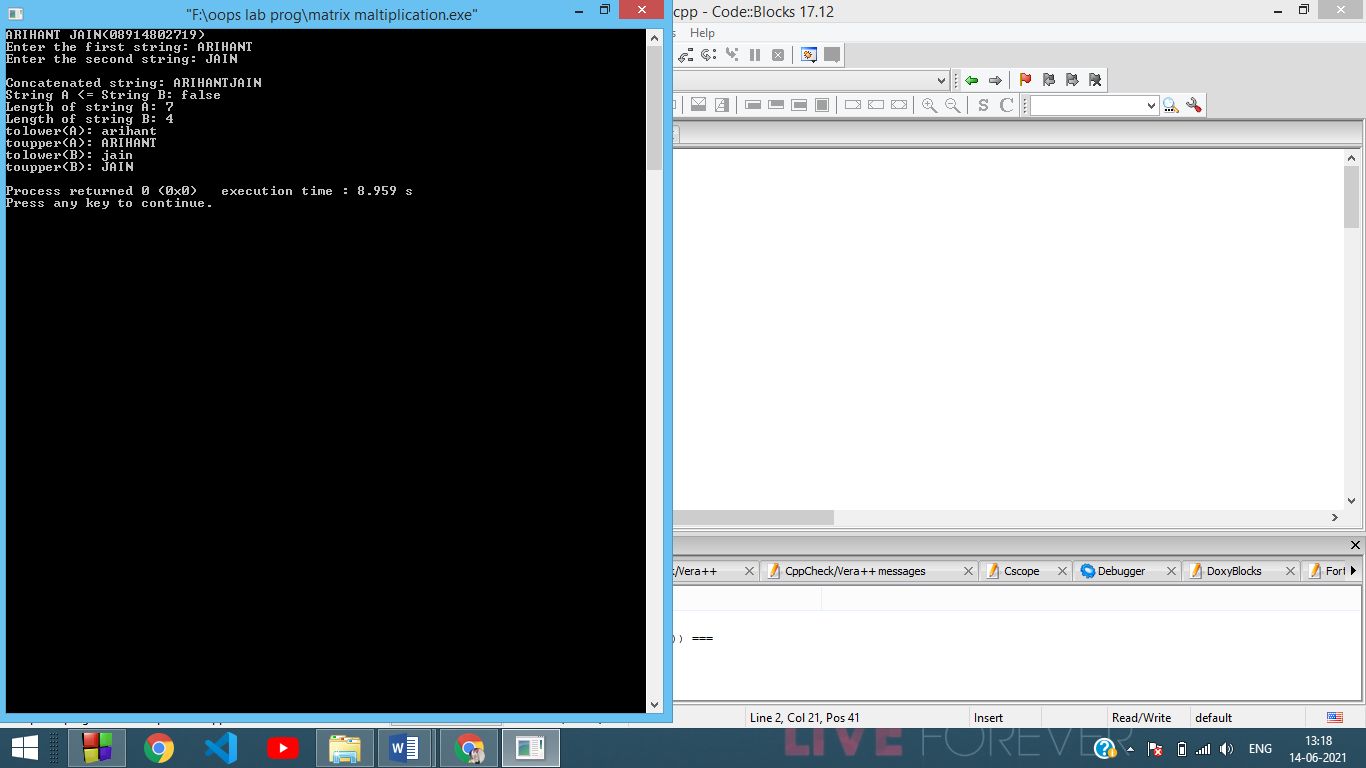
std::cout << "tolower(B): " << B\_lower.put\_string() << "\n";

std::cout <<"toupper(B): " << B\_upper.put\_string() << "\n";

return 0;

}

**OUTPUT:**

****

Experiment-5

**Aim:** To create a class called LIST with two pure virtual functions store () and retrieve (). To store a value call store and to retrieve, call retrieve function. Derive two classes stack and queue from it and override store and retrieve.

**Theory:** A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the function.

● Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

● They are mainly used to achieve Runtime polymorphism

● Functions are declared with a virtual keyword in base class.

● The resolving of function calls is done at Run-time.

Virtual functions cannot be static. A virtual function can be a friend function of another class. Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism. The prototype of virtual functions should be the same in the base as well as derived class. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used. A class may have a virtual destructor but it cannot have a virtual constructor.

**Code:**

#include <iostream>

using namespace std;

const int MAX=100;

int List[MAX];

int pos = 0;

// LIST Class

class LIST

{

public:

// Virtual function

virtual void store()

{

cout << "LIST class store() called.\n";

cout << "Enter the value: ";

cin >> List[pos++];

cout << "Value stored: " << List[pos-1] << "\n";

}

// Virtual function

virtual void retrieve()

{

cout << "LIST class retrieve() called.\n";

cout << "Enter the index: ";

int i;

cin >> i;

cout << "Value retrieved: " << List[i] << "\n";

}

};

// STACK Class

class STACK : public LIST

{

public:

// Overriden store() function

void store()

{

cout << "STACK class store() called.\n";

cout << "Enter the value: ";

cin >> List[pos++];

cout << "Value stored: " << List[pos-1] << "\n";

}

// Overriden retrieve() function

void retrieve()

{

cout << "STACK class retrieve() called.\n";

cout << "Enter the index: ";

int i;

cin >> i;

cout << "Value retrieved: " << List[i] << "\n";

}

};

// QUEUE Class

class QUEUE : public LIST

{

public:

// Overriden store() function

void store()

{

cout << "QUEUE class store() called.\n";

cout << "Enter the value: ";

cin >> List[pos++];

cout << "Value stored: " << List[pos-1] << "\n";

}

// Overriden retrieve() function

void retrieve()

{

cout << "QUEUE class retrieve() called.\n";

cout << "Enter the index: ";

int i;

cin >> i;

cout << "Value retrieved: " << List[i] << "\n";

}

};

int main()

{

// Create LIST base pointer

LIST \*list\_ptr;

// Create STACK and QUEUE objects

STACK s;

QUEUE q;

// Bind base pointer to derived class objects

list\_ptr = &s;

s.store();

q.store();

// Base class virtual function will not be called

list\_ptr->store();

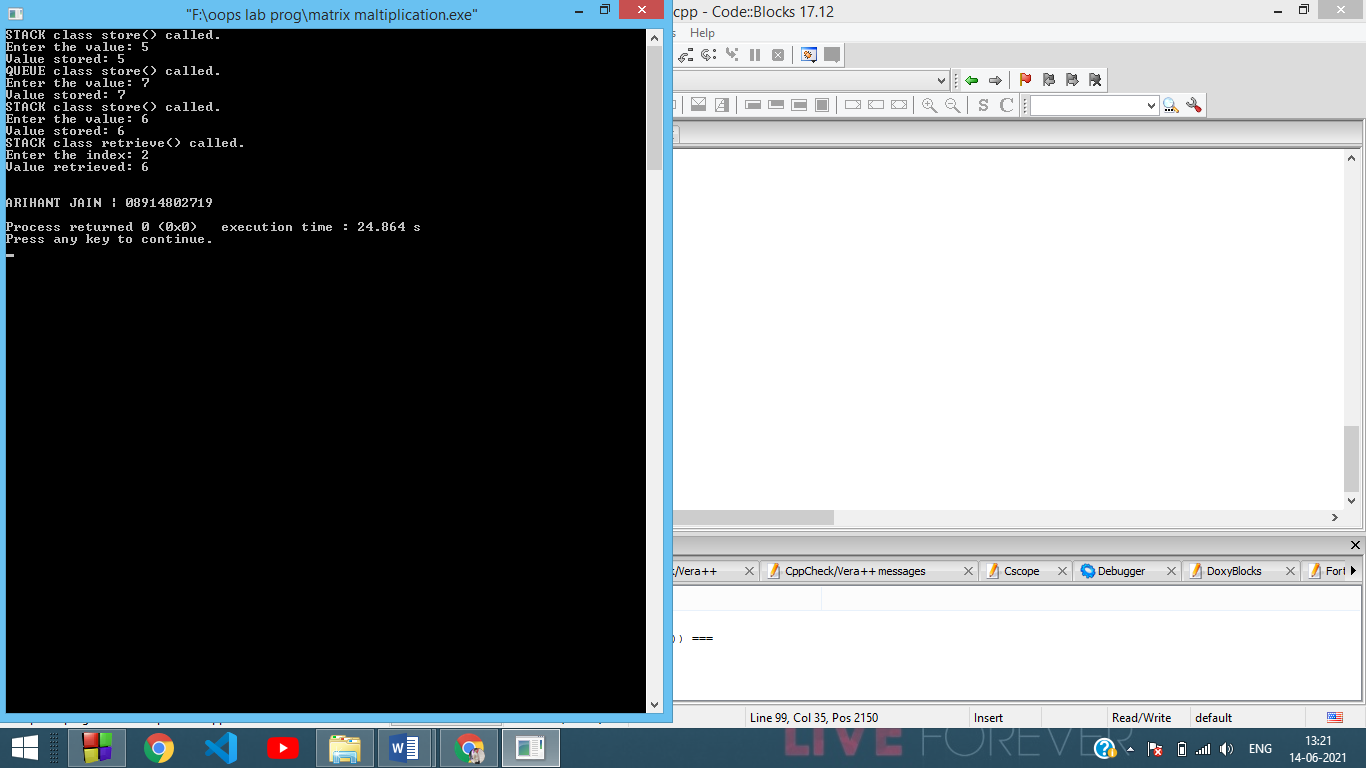
list\_ptr->retrieve();

cout << "\n\nARIHANT JAIN | 08914802719\n";

return 0;

}

**Output:**

****

Experiment-6

**Aim**: To write a program to define the function template for calculating the square of given numbers with different data types.

**Theory**: A **template** is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.

C++ adds two new keywords to support templates: ‘template’ and ‘typename’. The second keyword can always be replaced by the keyword ‘class’.

Templates are expanded at compile time. This is like macros. The difference is, the compiler does type checking before template expansion. The idea is simple: source code contains only function/class, but compiled code may contain multiple copies of the same function/class.

Like function templates, **class templates** are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

**Code**:

#include <iostream>

using namespace std;

// SQUARE function template

template <typename Type>

Type SQUARE (Type N)

{

Type square = N \* N;

return (Type) square;

}

int main()

{

// Initialise different data types

int a;

float b;

long c;

double d;

cout << "Enter int a: ";

cin >> a;

cout << "Enter float b: ";

cin >> b;

cout << "Enter long c: ";

cin >> c;

cout << "Enter double d: ";

cin >> d;

// Function templates being used

cout << "Square of int a: " << SQUARE(a) << "\n";

cout << "Square of float b: " << SQUARE(b) << "\n";

cout << "Square of long c: " << SQUARE(c) << "\n";

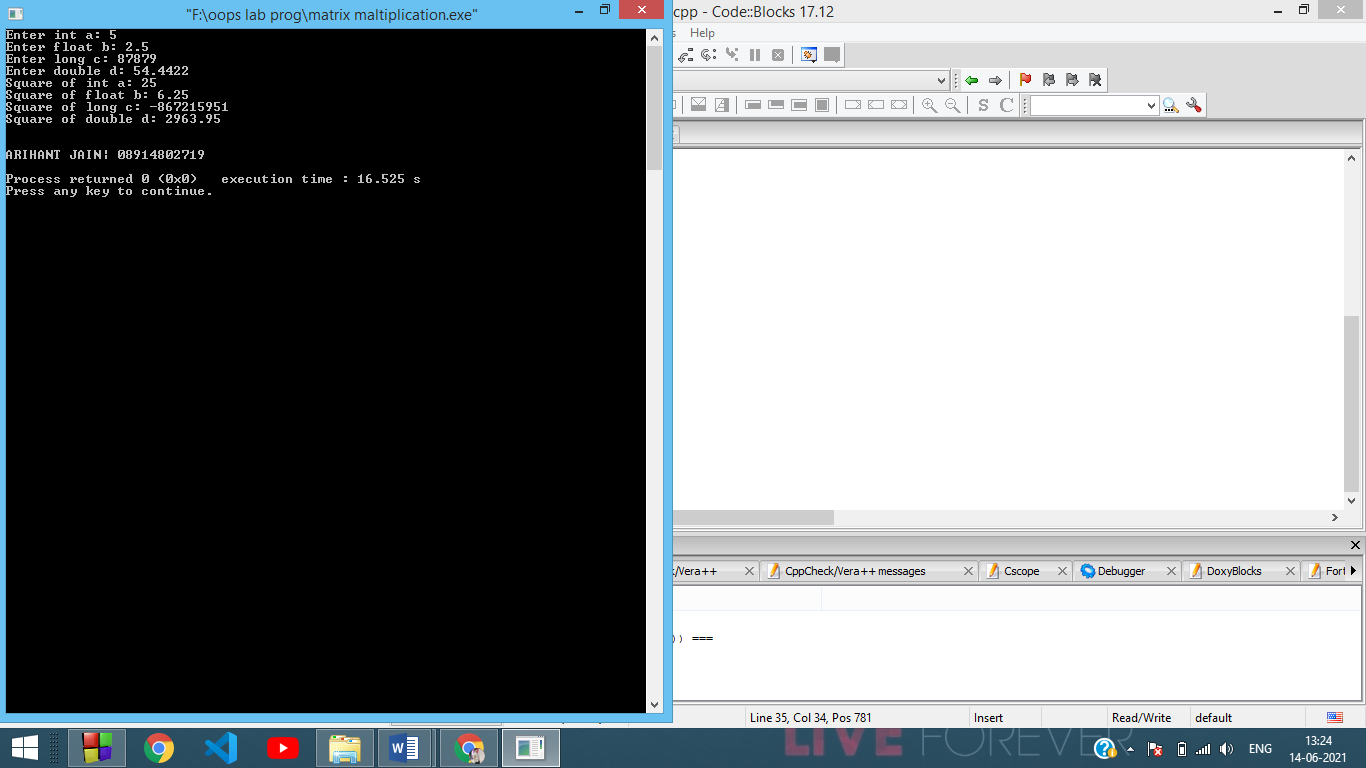
cout << "Square of double d: " << SQUARE(d) << "\n";

cout << "\n\nARIHANT JAIN| 08914802719\n";

return 0;

}

**Output**:



Experiment-7

**Aim**: To write a program to demonstrate the use of special functions, constructor and destructor in the class template. The program is used to find the bigger of two entered numbers.

**Theory**: A **constructor** is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. It can either accept the arguments or not. It is used to allocate the memory to an object of the class. It is called whenever an instance of the class is created. It can be defined manually with arguments or without arguments. There can be many constructors in class. It can be overloaded but it can not be inherited or virtual. There is a concept of copy constructor which is used to initialize an object from another object.

Like constructor, **destructor** is also a member function of a class that has the same name as the class name preceded by a tilde(~) operator. It helps to deallocate the memory of an object. It is called while an object of the class is freed or deleted. In a class, there is always a single destructor without any parameters so it can’t be overloaded. It is always called in the reverse order of the constructor. if a class is inherited by another class and both the classes have a destructor then the destructor of the child class is called first, followed by the destructor of the parent or base class.

**Code**:

#include <iostream>

using namespace std;

// Number class

class Number

{

int N;

public:

// Default constructor

Number()

{

N = 0;

}

// Parameterised constructor

Number(int X)

{

N = X;

}

int max(Number A)

{

if (N > A.N)

return N;

else

return A.N;

}

// Destructor

~Number()

{

cout << "Destructor working.\n";

}

};

int main()

{

int a, b;

cout << "Enter two numbers: ";

cin >> a >> b;

// Initialise two Number objects with constructor

Number A(a);

Number B(b);

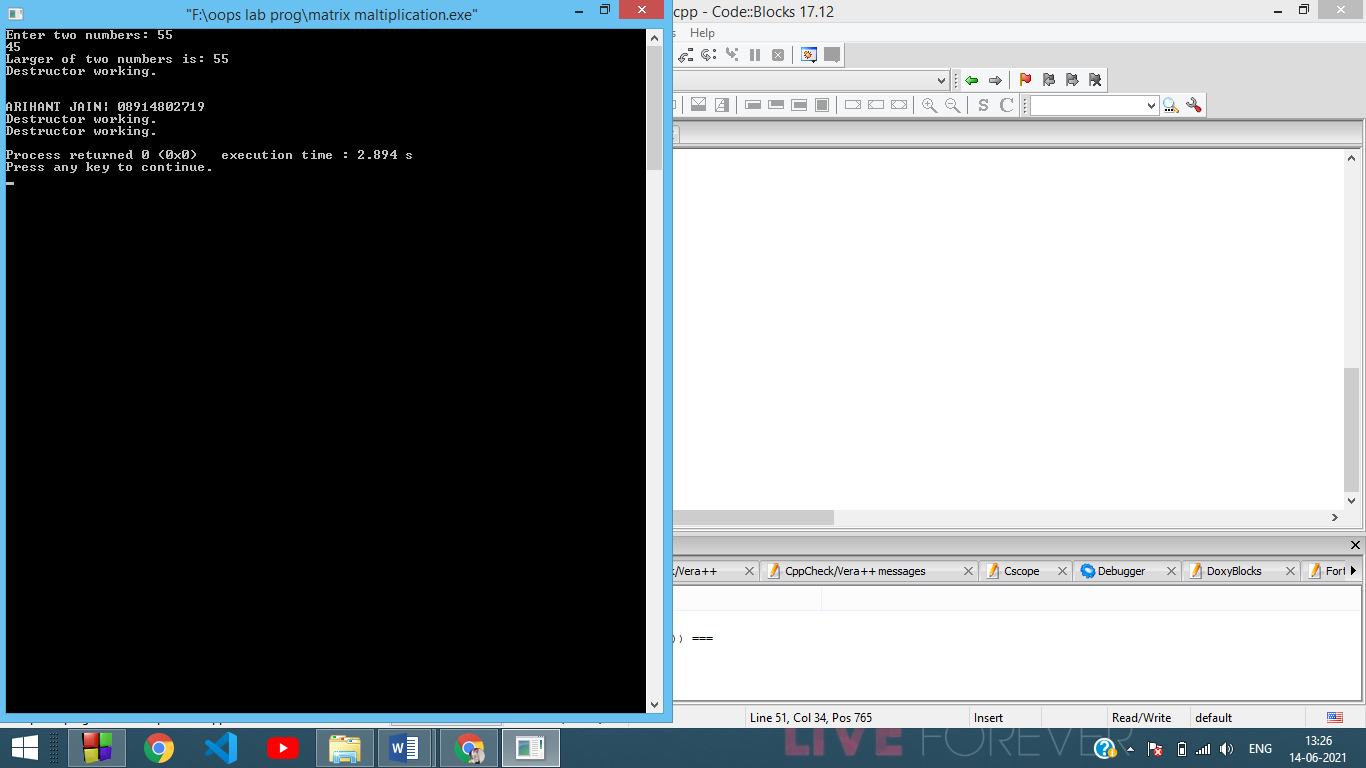
cout << "Larger of two numbers is: " << A.max(B) << "\n";

cout << "\n\nARIHANT JAIN| 08914802719\n";

return 0;

}

**Output**:



Experiment-8

**Aim**: To write a program to perform the deletion of white spaces such as horizontal tab, vertical tab, space, line feed, new line and carriage return from a text file and store the contents of the file without the white spaces on another file.

**Theory**: In C++, files are mainly dealt with using three classes: fstream, ifstream, ofstream available in fstream header file.

* **ofstream**: Stream class to write on files
* **ifstream**: Stream class to read from files
* **fstream**: Stream class to both read and write from/to files.

**Modes of opening**

* **in**: File open for reading: the internal stream buffer supports input operations.
* **out**: File open for writing: the internal stream buffer supports output operations.
* **binary**: Operations are performed in binary mode rather than text.
* **ate**: The output position starts at the end of the file.
* **app**: All output operations happen at the end of the file, appending to its existing contents.
* **trunc**: Any contents that existed in the file before it is open are discarded.

**Code**:

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

// Open input file

ifstream fi;

fi.open("input.txt");

// Open output file

ofstream fo;

fo.open("output.txt");

char ch;

// Traverse through file, char by char

// not skipping whitespaces

while (fi >> noskipws >> ch)

{

// Skip characters to be deleted

if (ch != '\t' and ch != '\v' and ch != ' ' and ch != '\n' and ch != '\r')

{

fo << ch;

}

}

fi.close();

fo.close();

// Open files for output

ifstream in;

in.open("input.txt");

ifstream out;

out.open("output.txt");

string line;

cout << "| INPUT FILE |\n\n";

while (in >> noskipws >> ch)

{

cout << ch;

}

cout << "\n\n| OUTPUT FILE |\n\n";

while (getline(out, line))

{

cout << line;

}

in.close();

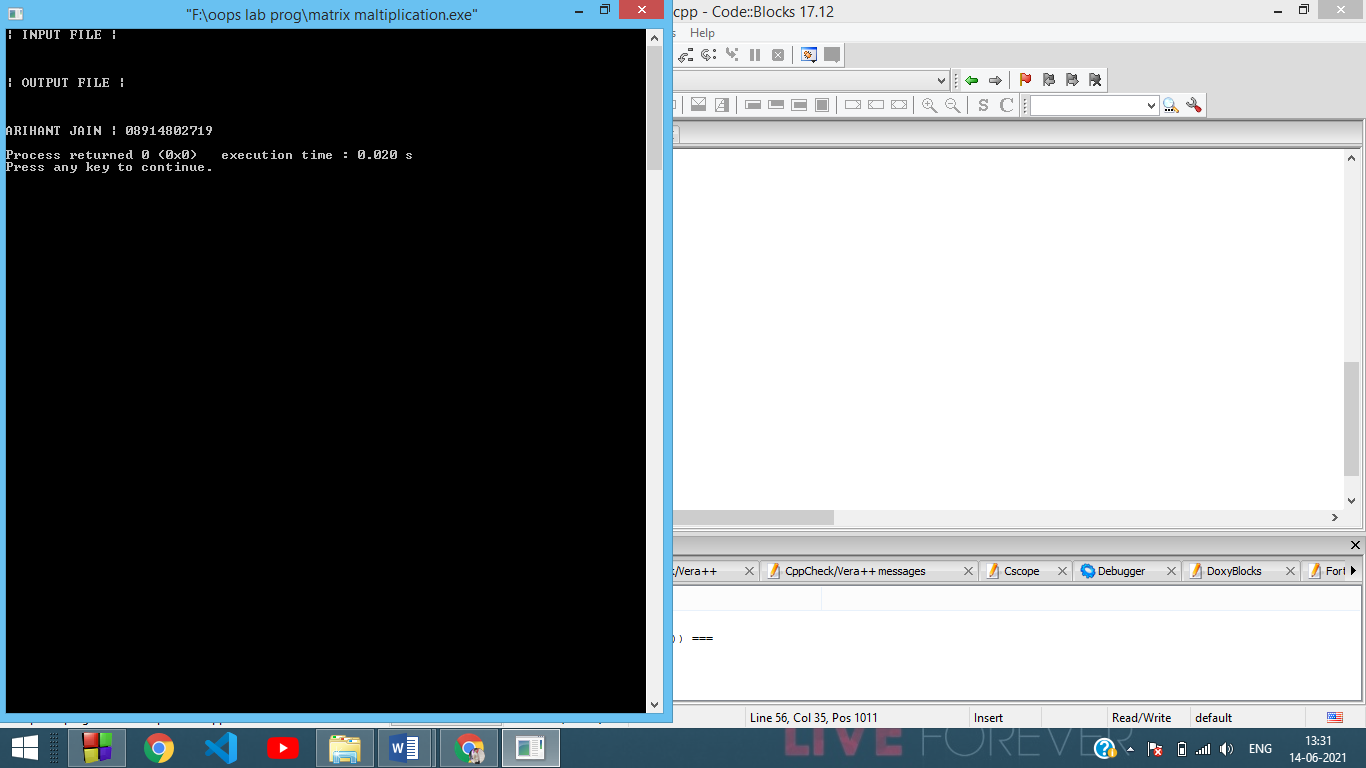
out.close();

cout << "\n\nARIHANT JAIN | 08914802719\n";

return 0;

}

**Output**:



Experiment-9

**Aim**: To write a program to read the class object of student info such as name, age, sex, height and weight from the keyboard and to store them on a specified file using read( ) and write( ) functions. Again, the same file is opened for reading and displaying the contents of the file on the screen.

**Code**:

#include <iostream>

#include <fstream>

using namespace std;

// Student class

class Student

{

string name;

int age;

char sex;

float height;

float weight;

public:

// Read student's data

void read()

{

cout << "Enter Name: ";

cin >> name;

cout << "Enter Age: ";

cin >> age;

cout << "Enter Gender (F/M): ";

cin >> sex;

cout << "Enter Height (metres): ";

cin >> height;

cout << "Enter Weight (kilogram): ";

cin >> weight;

}

// Write student's data in file

void write()

{

// Create file object

ofstream fi;

fi.open("data.txt");

string res = "Name: " + name + "\nAge: " + to\_string(age) + "\nGender: " + sex + "\nHeight: " + to\_string(height) + "m\nWeight: " + to\_string(weight) + "kg";

fi << res;

fi.close();

}

};

int main()

{

// Create Student object

Student S;

// Read student data

S.read();

// Write student data

S.write();

// Print file

ifstream fi;

fi.open("data.txt");

string line;

while (getline(fi, line))

{

cout << line << "\n";

}

cout << "\n\nHEMANT | 09314802719\n";

return 0;

}

**Output**:



Experiment-10

**Aim**: To write a program to raise an exception if any attempt is made to refer to an element whose index is beyond the array size.

**Theory**: One of the advantages of C++ over C is **Exception Handling**. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions:

* **Synchronous**
* **Asynchronous** (Ex:which are beyond the program’s control, Disc failure etc)

C++ provides the following specialized keywords for this purpose:

* **try**: represents a block of code that can throw an exception.
* **catch**: represents a block of code that is executed when a particular exception is thrown.
* **throw**: Used to throw an exception. Also used to list the exceptions that a function throws, but doesn’t handle itself.

If an exception is thrown and not caught anywhere, the program terminates abnormally. C++ library has a standard exception class which is the base class for all standard exceptions. All objects thrown by components of the standard library are derived from this class. Therefore, all standard exceptions can be caught by catching this type.

**Code**:

#include <iostream>

#include <exception>

using namespace std;

// Error structure definition

struct IndexOutOfBounds : exception

{

const char\* what() const noexcept

{

return "Array IndexOutOfBounds Error!\n";

}

};

int main()

{

// Input size of array

int size = 0;

cout << "Enter size of the array: ";

cin >> size;

// Create an array of size

int arr[size] = {0};

// Input elements

cout << "Enter elements of array: ";

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

cin >> arr[i];

// Search element by index

int pos;

cout << "Enter index to be searched: ";

cin >> pos;

try

{

// Throw error if index is out of bounds

if (pos >= size)

{

throw IndexOutOfBounds();

}

else

{

cout << "Element at index " << pos << ": " << arr[pos] << "\n";

}

}

// Catch and output error

catch (exception &e)

{

cout << e.what();

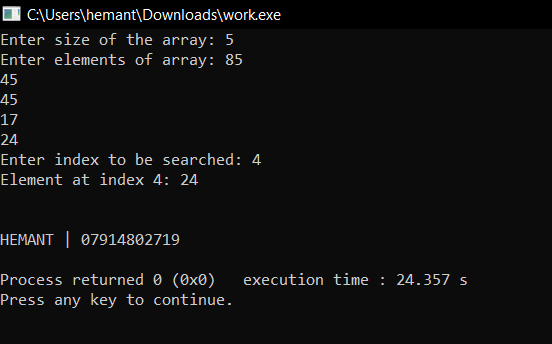
}

cout << "\n\nHEMANT | 07914802719\n";

return 0;

}

**Output**:



Experiment-11

**AIM:** To write a C++ program to find whether a number is prime or not.

**Source Code:**

#include <iostream>

using namespace std;

int main()

{

cout<<"ARIHANT JAIN(08914802719)"<<endl;

int n, i, m=0, flag=0;

cout << "Enter the Number to check Prime: ";

cin >> n;

m=n/2;

for(i = 2; i <= m; i++)

{

if(n % i == 0)

{

cout<<"Number is not Prime."<<endl;

flag=1;

break;

}

}

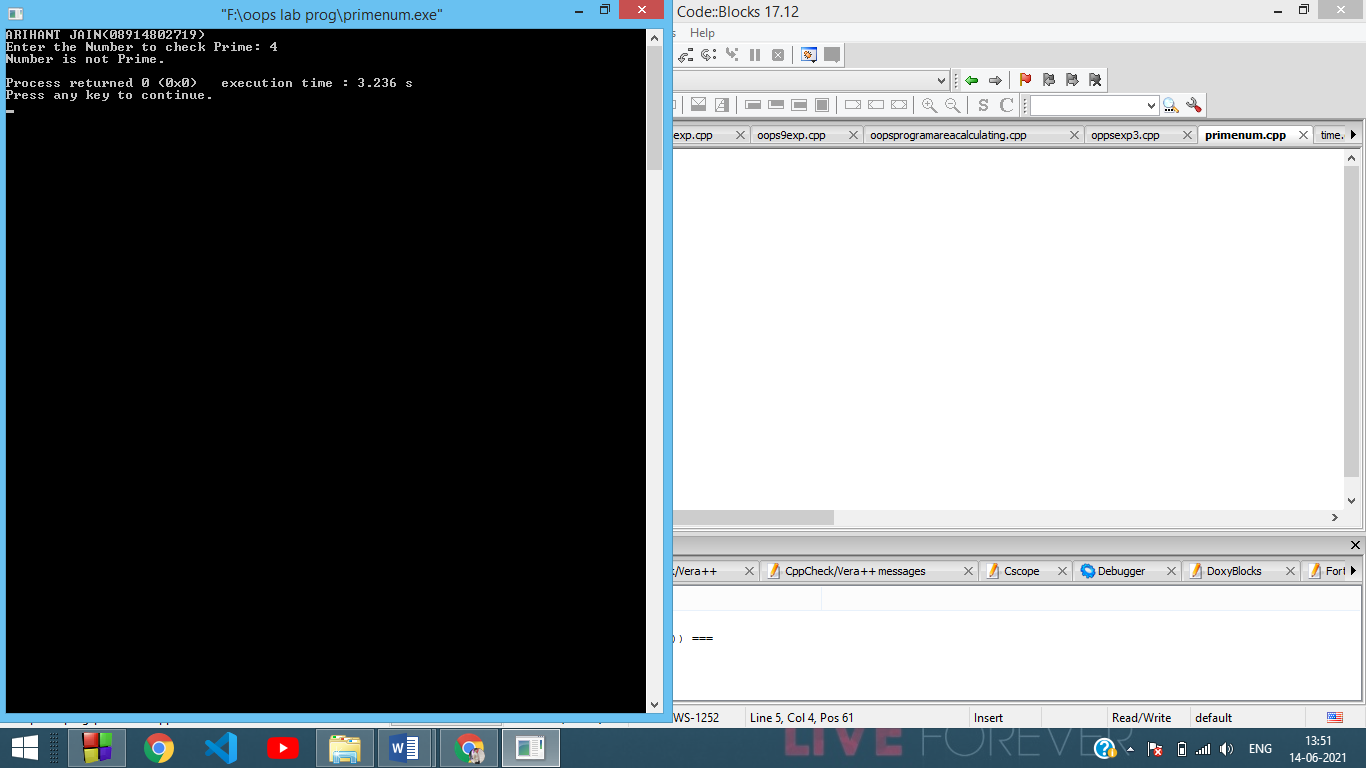
if (flag==0)

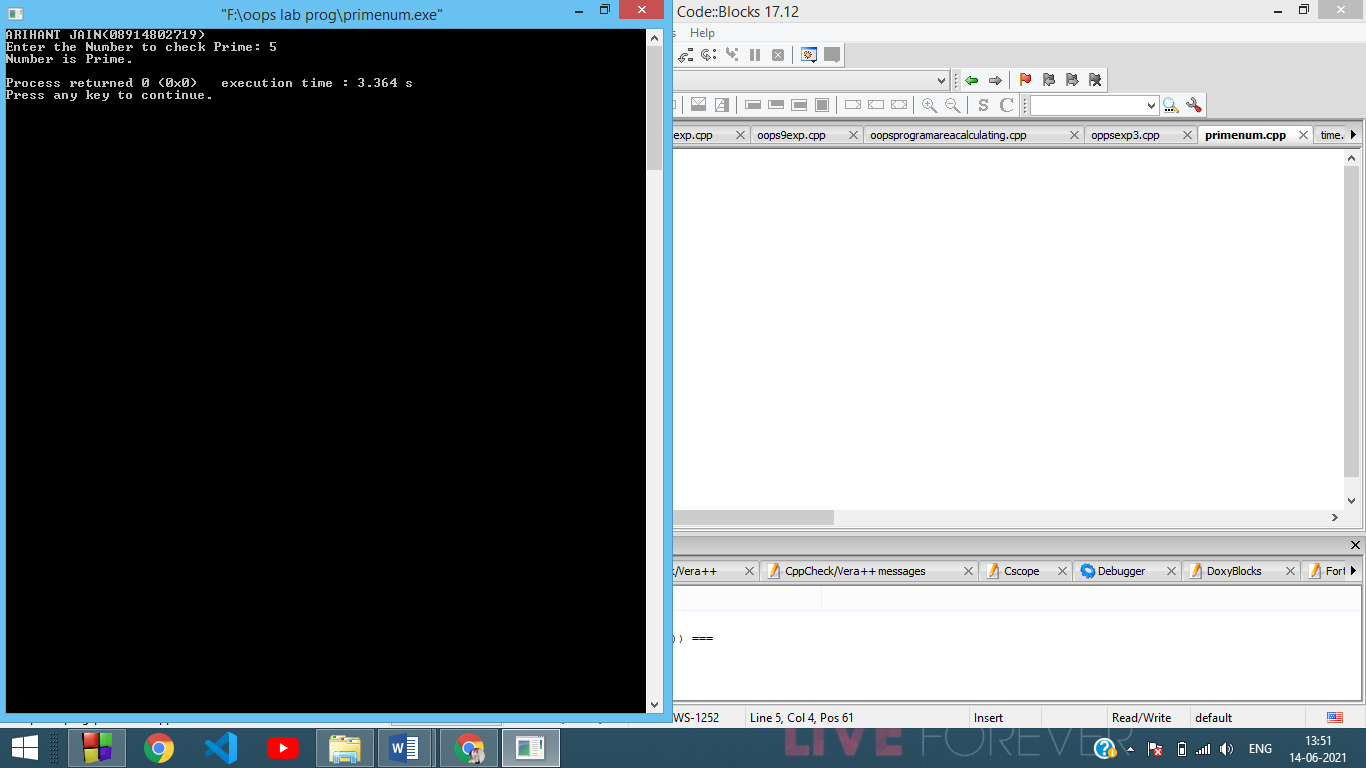
cout << "Number is Prime."<<endl;

return 0;

}

**OUTPUT:**

****

****

Experiment-12

**Aim**: To write a program to take name, address as character string, age as int, salary as float and contains an inline function to set the values and display it.

**Code**:

#include <iostream>

using namespace std;

#include<conio.h>

#include<stdio.h>

class employee

{

private:

char name[50];

char address[20];

int age;

float salary;

public:

void input()

{

cout<<"Enter Name:";

gets(name);

cout<<"Enter Address:";

gets(address);

cout<<"Enter Age:";

cin>>age;

cout<<"Enter Salary:";

cin>>salary;

}

void output();

};

void employee::output()

{

cout<<"Name:";

puts(name);

cout<<"Address:";

puts(address);

cout<<"Age:"<<age<<endl;

cout<<"Salary:"<<salary<<endl;

}

int main()

{

employee e;

cout<<"ARIHANT JAIN(08914802719)"<<endl;

cout<<"Enter the details==>"<<endl;

e.input();

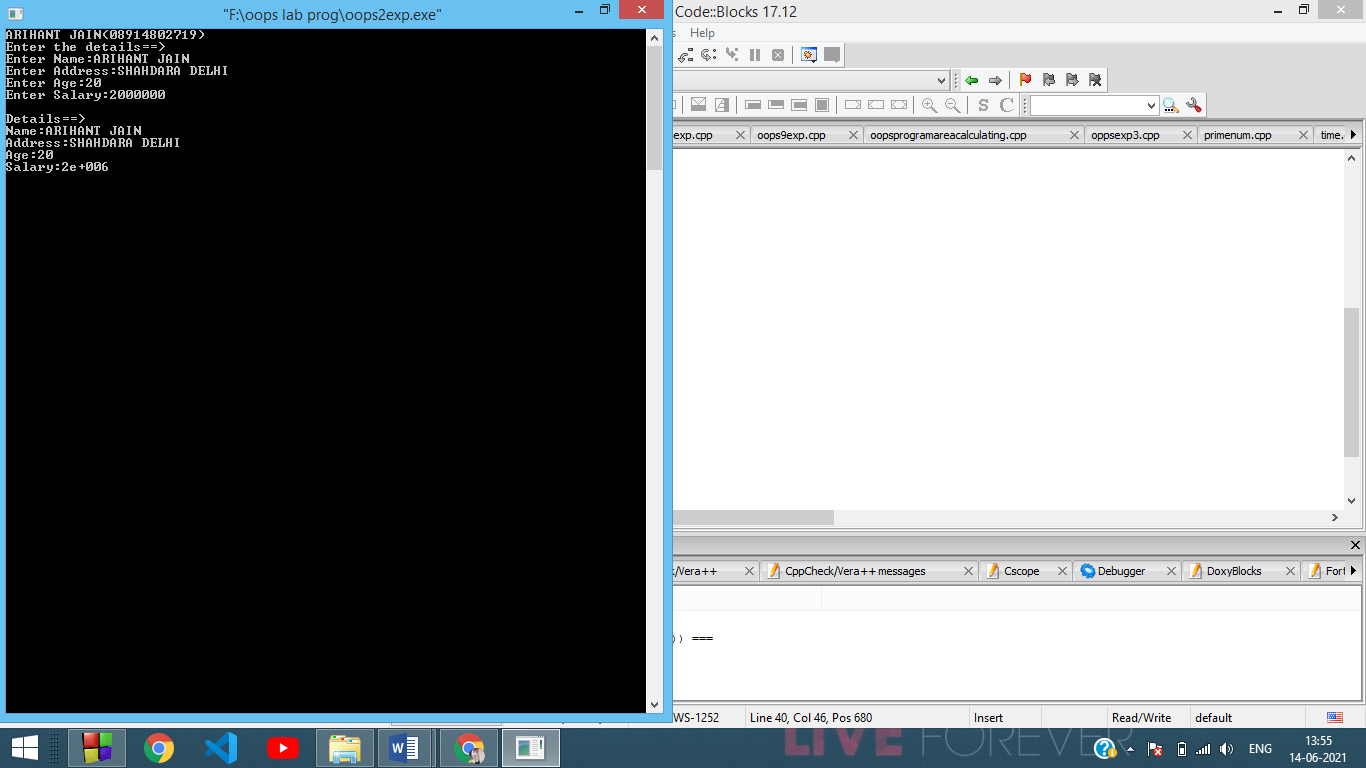
cout<<"\nDetails==>"<<endl;

e.output();

getch();

}

**Output**:



Experiment-13

**Aim**: Using the concept of function overloading, to write a function for calculating area of triangle, circle and rectangle.

**Code**:

#include<iostream>

using namespace std;

int area(int);

int area(int,int);

float area(float);

float area(float,float);

int main()

{

int s,l,b;

float r,bs,ht;

cout<<"ARIHANT JAIN(08914802719)"<<endl;

cout<<"Enter side of a square:";

cin>>s;

cout<<"Enter length and breadth of rectangle:";

cin>>l>>b;

cout<<"Enter radius of circle:";

cin>>r;

cout<<"Enter base and height of triangle:";

cin>>bs>>ht;

cout<<"Area of square is"<<area(s);

cout<<"\nArea of rectangle is "<<area(l,b);

cout<<"\nArea of circle is "<<area(r);

cout<<"\nArea of triangle is "<<area(bs,ht);

}

int area(int s)

{

return(s\*s);

}

int area(int l,int b)

{

return(l\*b);

}

float area(float r)

{

return(3.14\*r\*r);

}

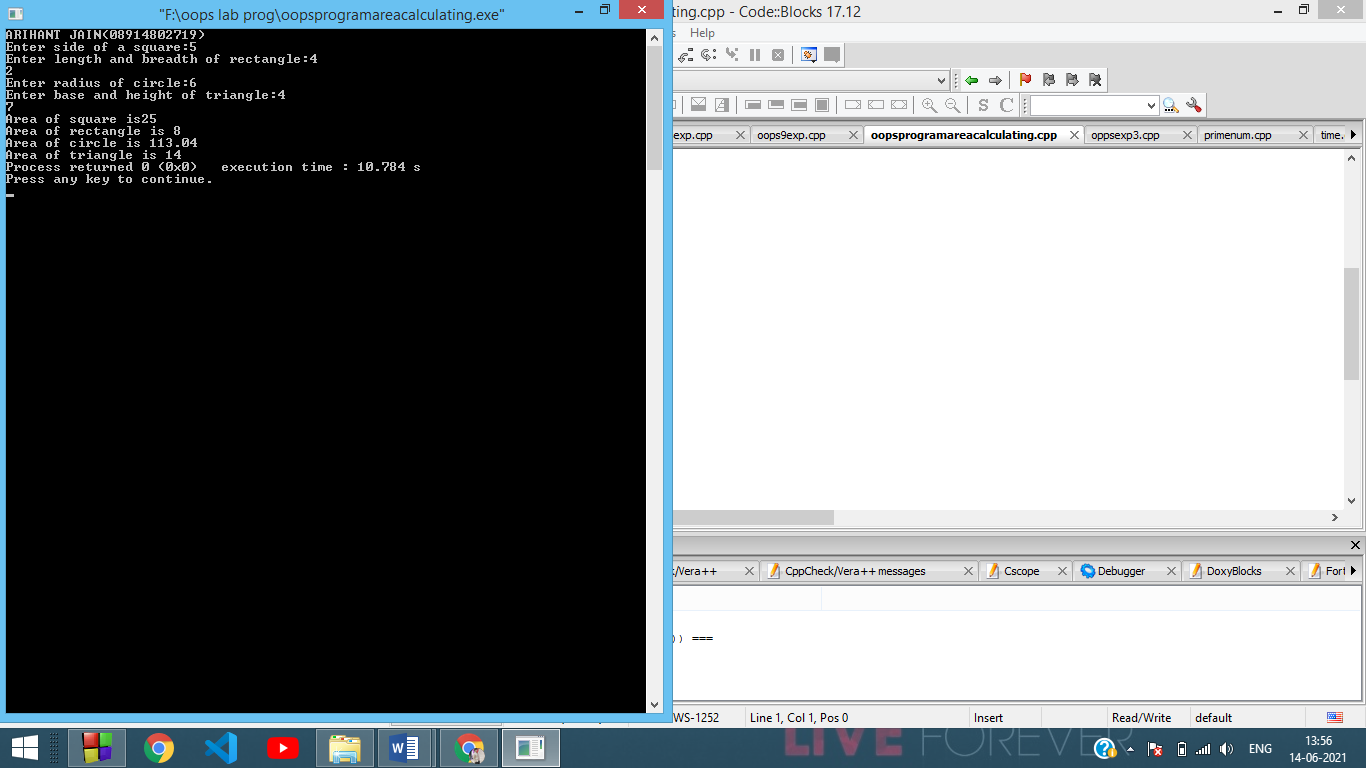
float area(float bs,float ht)

{

return((bs\*ht)/2);

}

**Output**:



Experiment-14

**Aim**: To create a class Student, which have data members as name, branch, roll no., age, sex, five subjects. Display the name of the student & his percentage who has more than 70%.

**Code**:

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**// Student class**

**class Student**

**{**

**string name;**

**string branch;**

**int roll;**

**int age;**

**char sex;**

**int marks[5];**

**float perc;**

**public:**

**// Read student data**

**void read()**

**{**

**cout << "\nEnter name: ";**

**cin >> name;**

**cout << "Enter branch: ";**

**cin >> branch;**

**cout << "Enter roll no.: ";**

**cin >> roll;**

**cout << "Enter age: ";**

**cin >> age;**

**cout << "Enter sex: ";**

**cin >> sex;**

**cout << "Enter marks in 5 subjects: ";**

**for (int i = 0; i < 5; i++)**

**cin >> marks[i];**

**}**

**// Calculate percentage**

**void calculate()**

**{**

**float total = 0;**

**for (int i = 0; i < 5; i++)**

**total += marks[i];**

**perc = total / 5;**

**}**

**// Print student info with percentage > 70%**

**void print()**

**{**

**if (perc > 70.0)**

**{**

**cout << "\nName: " << name << "\n";**

**cout << "Percentage: " << perc << "\n";**

**}**

**}**

**};**

**int main()**

**{**

**cout<<"ARIHANT JAIN(08914802719)"<<endl;**

**// Input student data size**

**int N;**

**cout << "Enter number of students: ";**

**cin >> N;**

**// Create Student array**

**Student S[N];**

**// Read Student data and calculate percentage**

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

**{**

**S[i].read();**

**S[i].calculate();**

**}**

**// Output student with percentage > 70%**

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

**{**

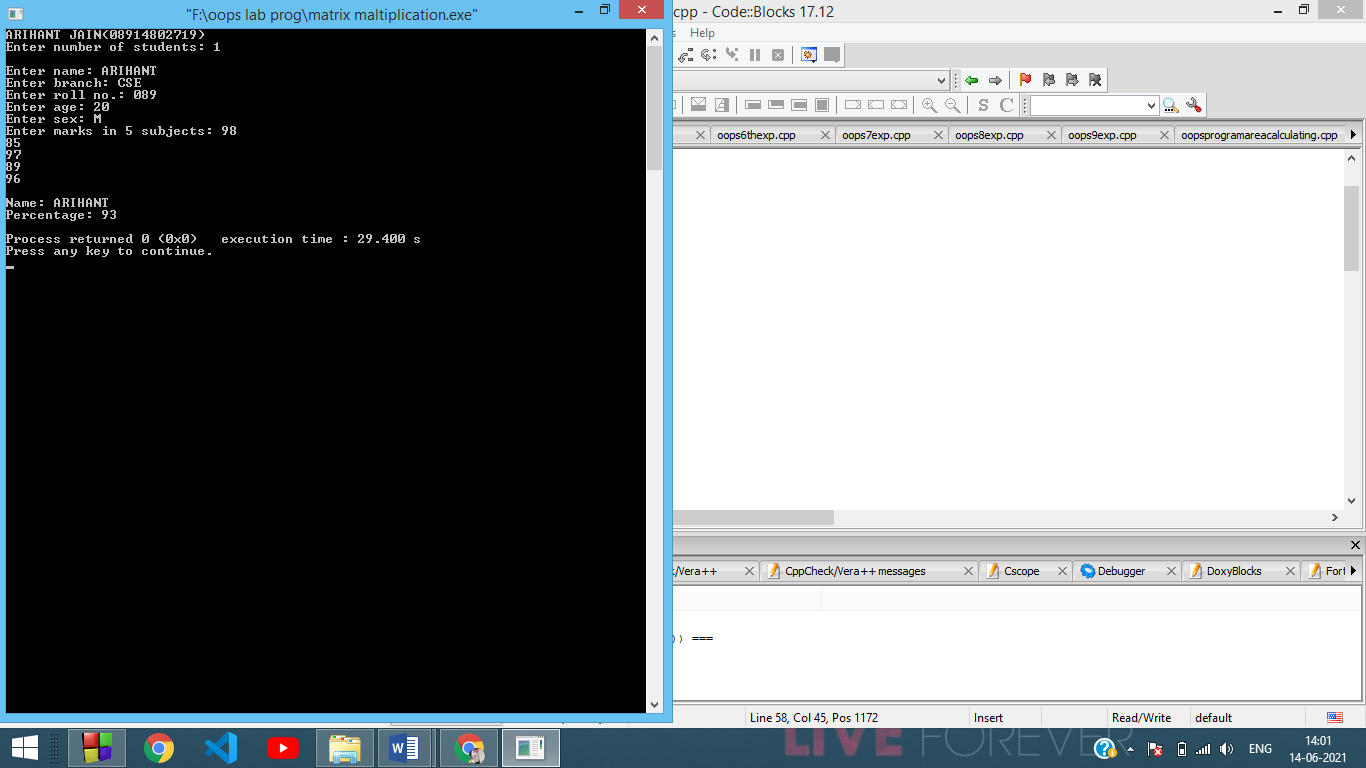
**S[i].print();**

**}**

**return 0;**

**}**

**Output**:



Experiment-15

**Aim**: To create a class TIME with members hours, minutes and seconds. Take input, add two-time objects and passing objects to function and display the result.

**Code**:

#include<iostream>

using namespace std;

class time{

int Hours;

int Minutes;

int Seconds;

public:

void getData(void);

void displayData(void);

void addData(time t1, time t2);

};

void time :: getData(void)

{

cout<<"enter time : "<<endl;

cout<<"enter hours"<<endl;

cin>>Hours;

cout<<"enter minutes "<<endl;

cin>>Minutes;

cout<<"enter seconds"<<endl;

cin>>Seconds;

}

void time:: displayData(void)

{

cout<<endl;

cout<<"Times after add:";

cout<<Hours<<":"<<Minutes<<":"<<Seconds<<endl;

}

void time:: addData(time t1, time t2)

{

this->Seconds=t1.Seconds+t2.Seconds;

this->Minutes=t1.Minutes+t2.Minutes + this->Seconds/60;;

this->Hours= t1.Hours+t2.Hours + (this->Minutes/60);

this->Minutes %=60;

this->Seconds %=60;

}

int main()

{

cout<<"ARIHANT JAIN(08914802719)"<<endl;

time t1,t2,t3;

t1.getData();

t2.getData();

//add two times

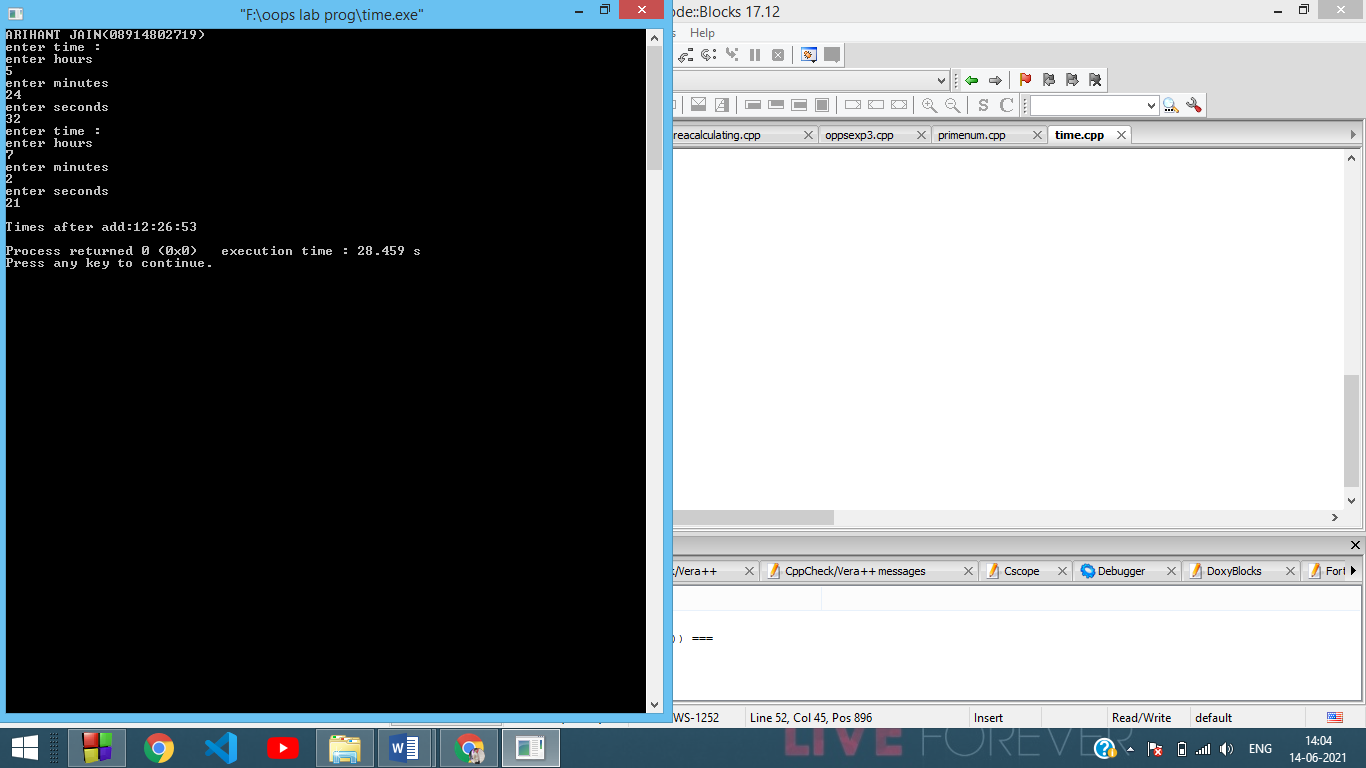
t3.addData(t1,t2);

t3.displayData();

return 0;

}

**Output**:



Experiment-16

**Aim**: To create a function power to raise a number m to a power n. The function takes a double value for m and integer value for n. Use the default value for n to make the function. Calculate the squares when this argument is omitted.

**Code**:

#include <iostream>

using namespace std;

#include<conio.h>

#include<math.h>

double power(double m,int n=2)

{

double t;

t=pow(m,n);

return t;

}

int main()

{

double num,ans;

int p;

cout<<"ARIHANT JAIN(08914802719)"<<endl;

cout<<"Enter the number: ";

cin>>num;

cout<<"enter the power:";

cin>>p;

if(p==0)

{

ans=power(num);

cout<<"Power of "<<num<<"is "<<ans<<endl;

}

else

{

ans=power(num,p);

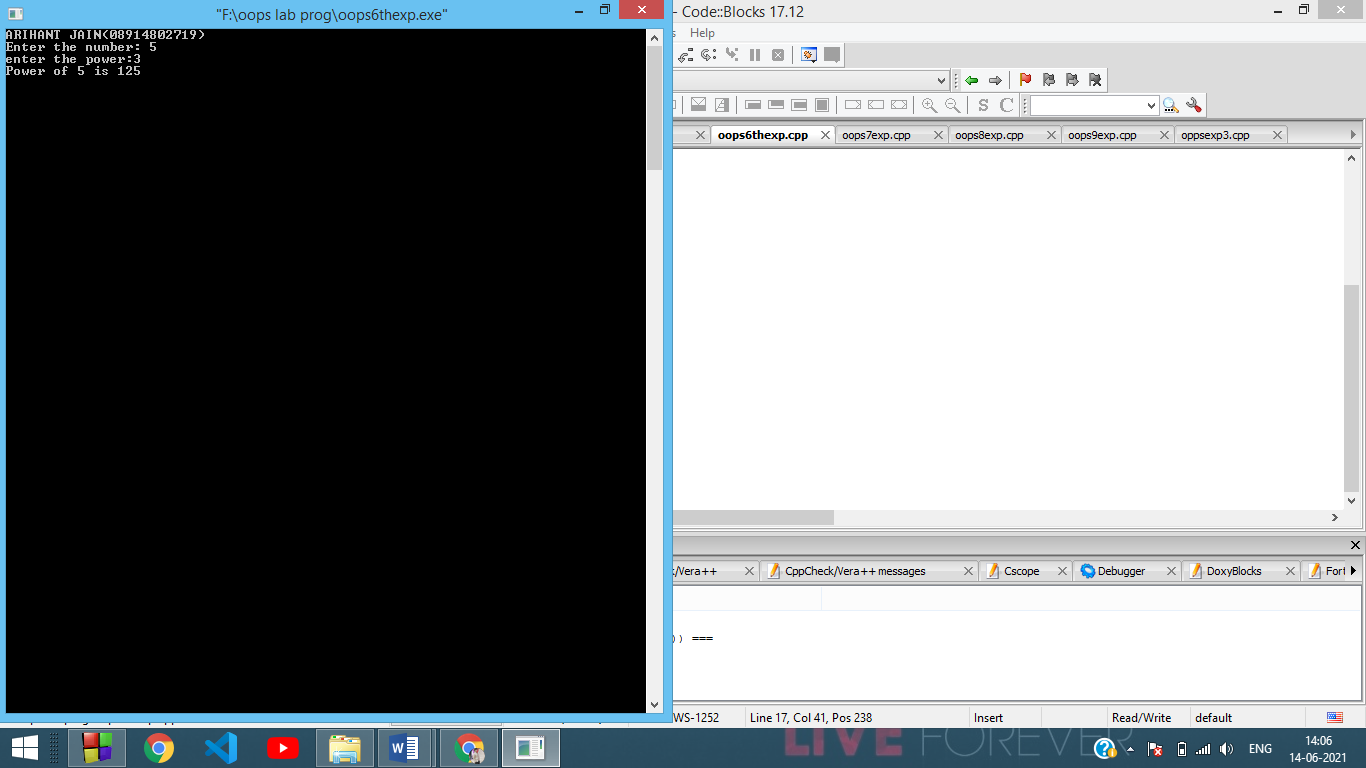
cout<<"Power of "<<num<<" is "<<ans<<endl;

}

getch();

}

**Output**:



Experiment-17

**Aim**: To write a program to enter any number and find its factorial using a constructor.

**Code**:

#include<iostream>

using namespace std;

class Test {

int n, i, f;

public:

Test() {

cout << "Enter a number:";

cin>>n;

f = 1;

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

f = f\*i;

}

}

void display() {

cout << "Factorial is:" << f;

}

};

int main() {

cout<<"ARIHANT JAIN(08914802719)"<<endl;

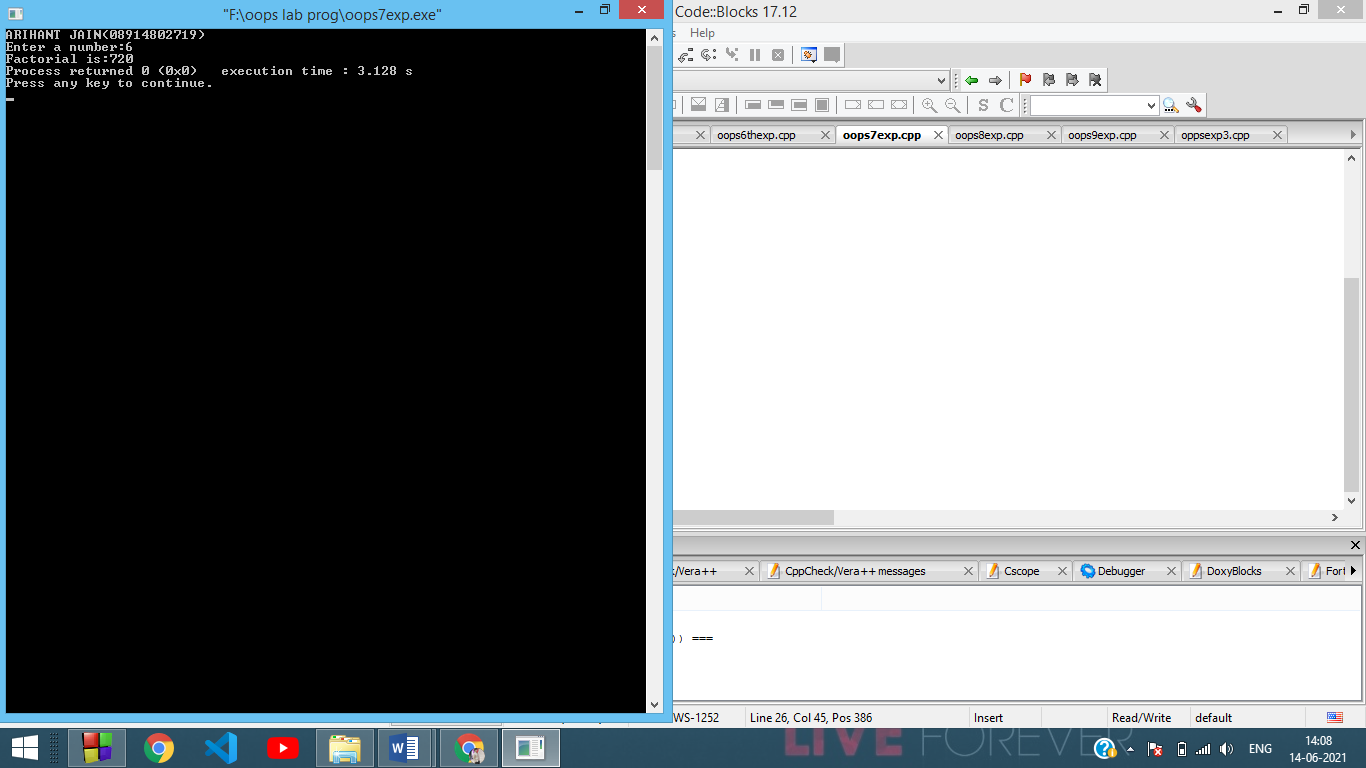
Test obj1;

obj1.display();

return 0;

}

**Output**:



Experiment-18

**Aim**: To write a program to generate a Fibonacci series using Copy Constructor.

**Code**:

#include<bits/stdc++.h>

using namespace std;

class fibonacci

{

private:

int a,b,c;

public:

fibonacci()

{

a=0;

b =1;

c=a+b;

}

fibonacci (fibonacci &ptr)

{

a=ptr.a;

b=ptr.b;

c=ptr.c;

}

void increment()

{

a=b;

b=c;

c=a+b;

}

void display()

{

cout << c << " ";

}

};

int main ()

{

cout<<"ARIHANT JAIN(08914802719)"<<endl;

int n;

cout<<"Enter the Number :";

cin>>n;

fibonacci f;

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

{

f.display();

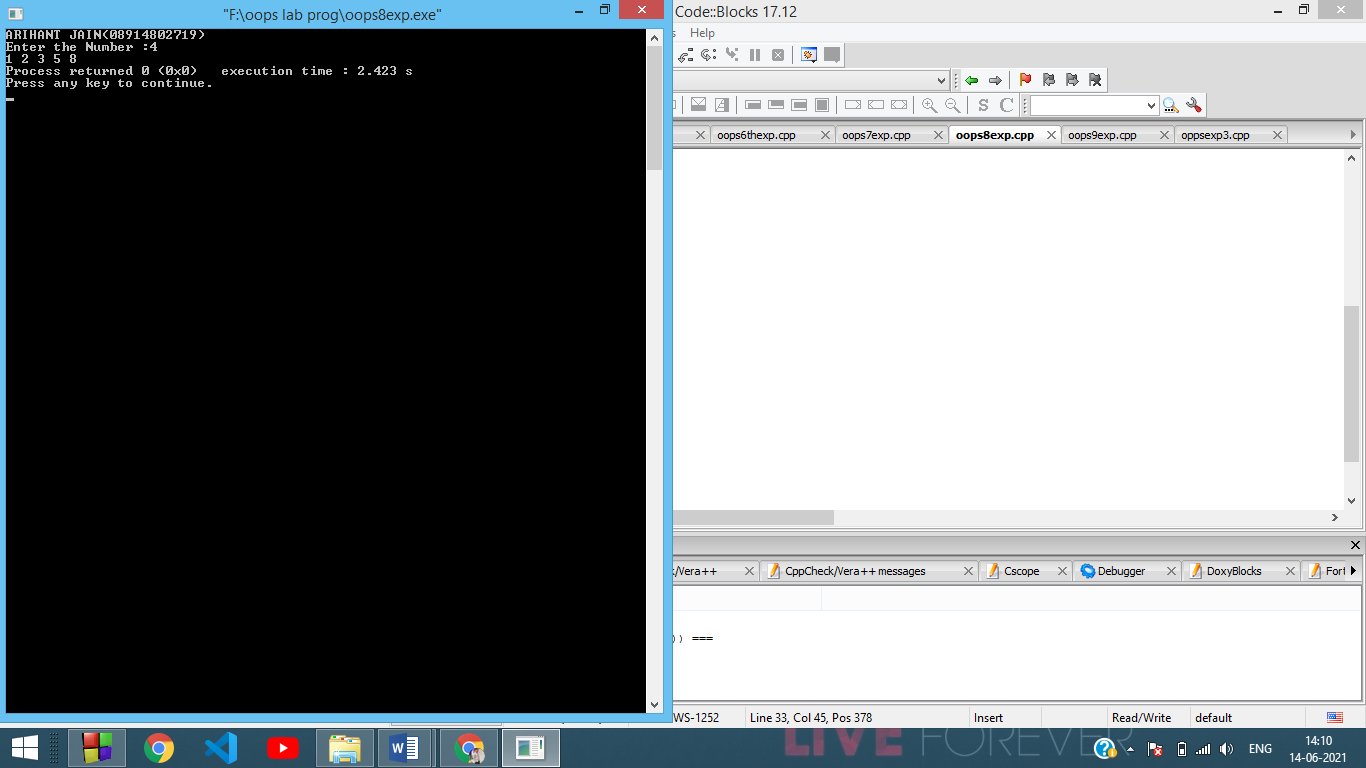
f.increment();

}

return 0;

}

**Output**:



Experiment-19

**Aim**: To write a program to find the biggest of three numbers using a Friend Function.

**Code**:

#include<iostream>

using namespace std;

class Test {

private:

int x, y, z;

public:

void input() {

cout << "Enter three numbers:";

cin >> x >> y>>z;

}

friend void find(Test t);

};

void find(Test t) {

if (t.x > t.y && t.x > t.z) {

cout << "Largest is:" << t.x;

} else if (t.y > t.z) {

cout << "Largest is:" << t.y;

} else {

cout << "Largest is:" << t.z;

}

}

int main() {

cout<<"ARIHANT JAIN(08914802719)"<<endl;

Test t;

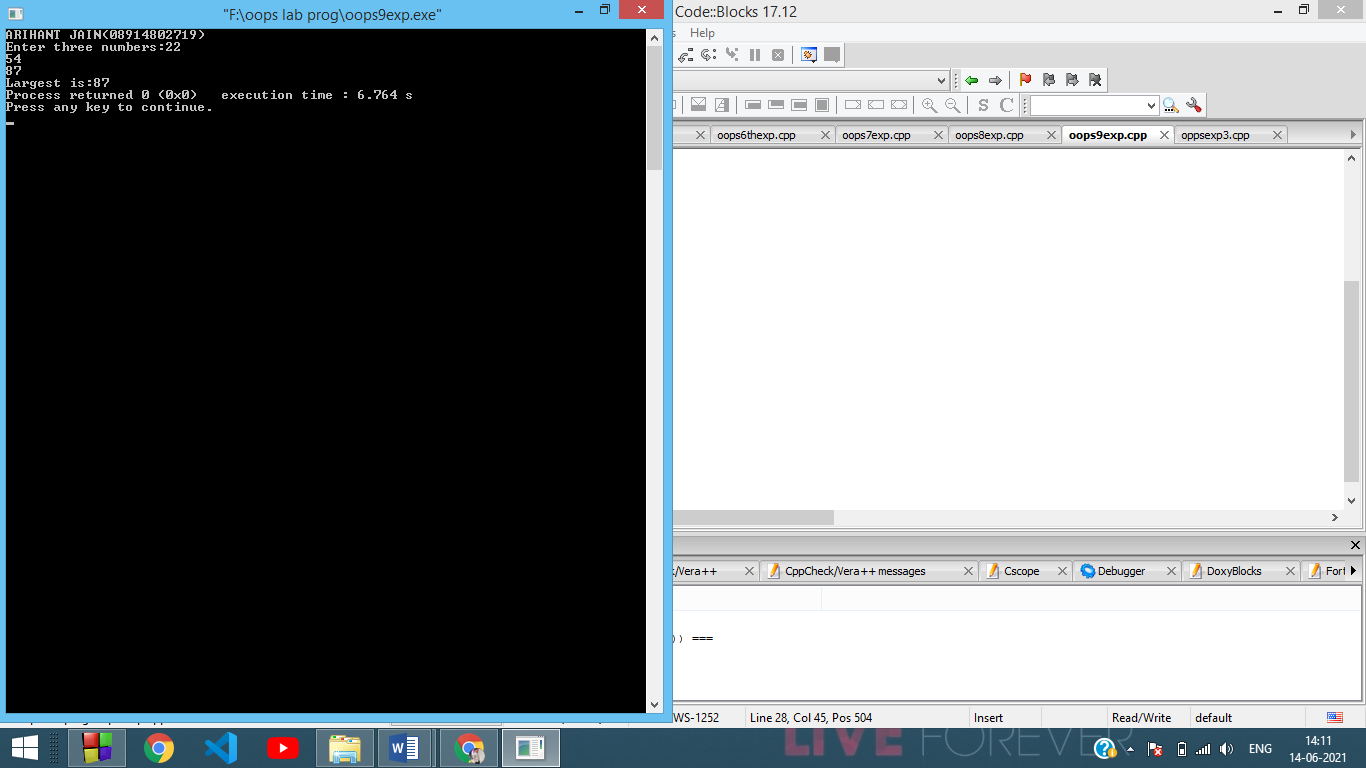
t.input();

find(t);

return 0;

}

**Output**:



Experiment-20

**Aim**: To write a program to demonstrate the use of a friend function with Inline assignment.

**Code**:

#include <iostream>

using namespace std;

// Number class

class Number

{

int N;

public:

// Friend function to read N

inline friend void read(Number&);

// Friend function to print N

inline friend void print(Number&);

};

void read(Number &X)

{

cout << "Enter a private number: ";

cin >> X.N;

cout << "Private value successfully entered\n\n";

}

void print(Number &X)

{

cout << "Private value: " << X.N << "\n";

}

int main()

{

cout<<"ARIHANT JAIN(08914802719)"<<endl;

// Create Number object

Number A;

// Read private member value through friend function read()

read(A);

// Print private member value through friend function read()

print(A);

return 0;

}

**Output**:

