Savitribai Phule Pune University

S. Y. B. B. A. (C. A.) Semester-IV
(CBCS 2019 Pattern)

Object Oriented Concepts Through CPP, NODE JS and Advance PHP

CA-406: Lab Book

Roll No.: __________ Division: ______ Seat No. :________________________

Student Name: ____

College Name:____

Academic Year:_________________________________

CERTIFICATE

This is to certify that Mr./Ms.________________________________ Seat number ________ of S.Y.B.B.A. (C.A) Sem-IV has successfully completed Laboratory Course (Object Oriented Concepts Through CPP and NODE JS / Advance PHP) in the year _______________. He/She has scored _____mark out of 10 (for Lab book).

Subject Teacher	**H.O.D./Coordinator**
Internal Examiner	**External Examiner**

Introduction

1. **About the work book:**

   This workbook is intended to be used by S.Y.B.B.A. (C.A.) Semester-IV students for Object Oriented Concepts Through CPP, NODE JS and Advance PHP assignments. It is designed by considering all the practical topics mentioned in the syllabus.

2. **The objectives of this workbook are:**

   - Defining the scope of the course.
   - To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
   - To have continuous assessment of the course and students.
   - Providing ready reference for the students during practical implementation.
   - Provide more options to students so that they can have good practice before facing the examination.
   - Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. **How to use this workbook:**

   The workbook is divided into three sections. Section-I is related to CPP assignments, Section-II is related to NODE JS assignments and Section-III is related to Advance PHP assignments.
   Section-I CPP is divided into nine assignments.
   Section-II NODE JS is divided into four assignments.
   Section-III Advance PHP is divided into six assignments.
   From Section-II and Section-III students have to perform practical assignments of selected elective subject only.
   Each assignment of all sections has three SETs-A, B and C. It is mandatory for students to complete SET A and SET B in lab. Assignment also includes practice programs which are expected to be solved by students as home assignments and to be evaluated by subject teachers.

4. **Instructions to the students**

   Please read the following instructions carefully and follow them during practical.
   - Students are expected to carry this workbook every time they come to the lab for computer practical.
   - Students should prepare for the assignment by reading the relevant material which is mentioned in ready reference.
   - Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to cover all workbook assignments if needed.

- Students will be assessed for each assignment on a scale from 0 to 5.

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

5. Instruction to the Instructors

Make sure that students should follow above instructions.

- Explain the assignment and related concepts in around ten minutes using whiteboard if required or by demonstrating the software.
- Evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion sheet of the respective section.

6. Instructions to the Lab administrator

You have to ensure appropriate hardware and software is made available to each student.
The operating system and software requirements on server side and also client side areas given below:

- Operating System - Windows
- Turbo C++
- Wamp Server
- Visual Studio Code

Assignment Completion Sheet

Section-I: Object Oriented Concepts Through CPP			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Beginning with C++		
2	Operators and Functions in C++		
3	Classes and Objects		
4	Constructors and Destructors		
5	Inheritance		
6	Polymorphism		
7	Managing Console I/O operations		
8	Working with Files		
9	Templates		
	Total (Out of 45)		
	Total (Out of 5)		

Instructor Signature:

Section-II: NODE JS			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Node.js web server, modules & npm		
2	File system		
3	Events in node.js		
4	Node.js with database		
Total (Out of 20)			
Total (Out of 5)			

'OR'

Section-III: Advance PHP			
Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Introduction to Object Oriented Programming in PHP		
2	To study Web Techniques		
3	XML		
4	PHP with AJAX		
5	Connecting Database using PHP & AJAX		
6	PHP Framework - Druple		
Total (Out of 30)			
Total (Out of 5)			

Instructor Signature:

Section-I

Object Oriented Concepts Through CPP

Assignment No. 1: Beginning with C++

Introduction:

In 1982, Bjarne Stroustrup started to develop a successor to C with Classes at Bell labs, which he named "C++", as it is an extension to C programming language. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.
Object-oriented programming has several following advantages over procedural programming:

- OOP is faster and easier to execute.
- OOP provides a clear structure for the programs.
- OOP makes the code easier to maintain, modify and debug.
- OOP makes it possible to create full reusable applications with less code and shorter development time.
- OOP makes development and maintenance easier if code grows as project size grows.
- OOP provide data hiding
- OOP provide ability to simulate real-world event much more effectively.

C++ is a general purpose, object oriented programming language. C++ has some additional facilities to those in C such as classes, data binding, data hiding, inheritance, encapsulation, polymorphism, default function argument etc. because of which it allows code to be reused and lowering development costs.

Real-World applications of C++:

- C++ is close to the hardware, can easily manipulate resources and it is fast, which makes it a primary choice to develop the **gaming systems**.

- C++ can be used to develop most of the **GUI based and desktop applications** easily. Example: Adobe Photoshop, Win amp media player from Microsoft.

- C++ is also used in writing database management software. The two most popular databases **MySQL and Postgres** are written in C++.

- The fact that C++ is a strongly typed and fast programming language makes it an ideal candidate for writing **operating systems**. Apple OS X has some of its parts written in C++. Similarly, some parts of the iPod are also written in C++. Most of the software from Microsoft is developed using C++ (flavors of Visual C++). Applications like Windows 95, ME, 98; XP, etc. are written in C++. Apart from this, the IDE Visual Studio, Internet Explorer and Microsoft Office are also written in C++.

- **Browsers** are mostly used for rendering purposes. Rendering engines need to be faster in execution as most people do not like to wait for the web page to be loaded. With the fast performance of C++, most browsers have their rendering software written in C++.Mozilla Firefox internet browser is an open-source project and is developed completely in C++.Google applications like Google File System and Chrome browser are written in C++.

- C++ is useful in developing an **application that requires high-performance image processing, real-time physical simulations, and mobile sensor applications** that need high performance and speed. Maya 3D software from Alias system is developed in C++ and is used for animation, virtual reality, 3D graphics, and environments.

- **Compilers** of various high-level programming languages are written either in C or C++. The reason is that both C and C++ are low-level languages that are close to hardware and are able to program and manipulate the underlying hardware resources.

- C++ can be used for building higher-level applications with graphics libraries, **applications to communicate with network devices** and computer network simulators as well as remote device systems and network management.

C++ Data types:

Data types in C++ are mainly divided into three types:

1. **Primitive/Built-in Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. Primitive data types available in C++ are:
   - int
   - char
   - bool
   - float
   - double
   - void
2. **Derived Data Types:** The data-types that are derived from the primitive or built-in data types are referred to as Derived Data Types. These can be of four types namely:
   - Function
   - Array
   - Pointer
   - Reference
3. **Abstract or User-Defined Data Types**: These data types are defined by user itself. C++ provides the following user-defined data types:
   - Class
   - Structure
   - Union
   - Enumeration
   - Typedef defined DataType

Simple C++ Program:

Example: C++ Hello world program to simply print "Hello World" on computer screen.

```
// My first C++ program
#include <iostream.h>
int main()
{
        cout<< "Hello World!";
        return 0;
}
```

Fundamental components in C++ programs:

- **Comments: // My first C++ program**
  In above C++ program first line with double slash symbol indicate single line comment and to indicate multiline comment programmer can enclose multiple lines in /*   */ which means these lines inserted by the programmer has no effect on the behavior of the program. Programmers use comment to include short explanations or observations about program.

- **Header files: #include <iostream.h>**
  Lines beginning with '#' are directives which are read and interpreted by preprocessor before the compilation of the C++ program begins. In above C++ program directive #include <iostream.h>, instructs the preprocessor to include header file iostream, which allows to add the contents of the iostream.h file to the program to perform standard input and output operations, such as accepting input through keyboard and writing the output of program on screen.

- **Main function: int main ()**
  The function named main is a special function in all C++ programs; it is the function which called by operating system automatically when C++ program run. The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.
  Proper way of writing the main function in C++ is to use int return type for main function. C++ standards and specifications mention that the main function should always return integer value which can be '0' or '1' where '0' is the standard for "successful execution of the program".

- **Operator:**
  In C++ input and output are performed in the form of a sequence of bytes or more commonly known as streams. Streams are of two types, if the direction of flow of bytes is from the device like keyboard to main memory then it is an Input Stream and if the direction of flow of bytes is from main memory to device like display screen then it is an output stream.

C++ is able to input and output the built-in data types using the stream extraction operator >> and the stream insertion operator << respectively. iostream stands for standard input-output stream in C++, this header file contains definitions to objects like cin, cout etc.

C++ Input and Output Operators:
In C++ cout is a predefined object which is an instance of ostream class. The cout object is connected to the standard output device, which usually is the display screen. The cout is used in conjunction with output operator "<<" ("put to"), also known as stream insertion operator to direct a value to standard output i.e. to display output on the screen.
cin is a predefined object which is an instance of istream class. The cin object is connected to the standard input device, which usually is the keyboard. The cin is used in conjunction with input operator ">>" ("get from"), also known as stream extraction operator to read a value from standard input i.e. to accept input from keyboard.

Example: C++ program to illustrate use of input and output operators.
```
#include<iostream.h>
int main()
{
  int num1, num2;
  cout<<"\n Enter two numbers :";
  cin>>num1>>num2;                                   //cascaded input operators
  cout<<"\n Two numbers are:"<<num1<<"   "<<num2;    // cascaded output operators
  return 0;
}
```

Cascading of Input / Output Operators in C++:
The cascading of the input and output operators refers to the multiple use of input or output operators in one statement. The statement using multiple output operators "<<" is said to be cascading output operator and the statement using multiple input operators ">>" is said to be cascading input operator.
In above program, the cascaded input operators wait for the user to input two integer values, where values are assigned from left to right, means first input value will get assigned to num1 and second input value will get assigned to num2. The cascaded output operator first displays the message "Two numbers are:", then displays the value ofnum1 and after that it displays the value of num2.
It is observed that cascading of the input/output operator improves the readability and reduces the size of the program.

- **C++ Program Execution:**
  Create a C++ program using editor in Turbo C++.Save the program using F2, give a meaningful name to a source file which should reflects the purpose of the program, with extension ".cpp". Compile the program using Alt + F9.Execute your C++ program by pressing Ctrl + F9. Press Alt + F5 to view the output of the program at the output screen.

Practice Programs:

1. Write a C++ program to find factorial of a given number.
2. Write a C++ program to check whether a given number is even or odd.
3. Write a C++ program to check whether a given number is prime or not.
4. Write a C++ program to check whether a given number is perfect or not.
5. Write a C++ program to find largest and smallest number of 3 integer numbers. (Use cascading of I/O operators.)

Set A:

1. Write a C++ program to generate multiplication table.
2. Write a C++ program to display first ‘n’ numbers of Fibonacci series.
3. Write a C++ program to reverse a number.
4. Write a C++ program to display Armstrong numbers between two intervals.
5. Write a C++ program to accept two integers and an arithmetic operator(+, -, *, /) from user and performs the corresponding arithmetic operation and display the result. (Use switch statement)

Set B:

1. Write a C++ program to print the following pattern.
```
A
B   C
D   E   F
G   H   I   J
```

2. Write a C++ program to print the following pattern
```
         *
      *  *
   *  *  *
*  *  *  *
```

3. Write a C++ program to calculate following series:
   (1*1)+(2*2)+(3*3)+ … + (n*n)

4. Write a C++ program to convert decimal number into binary number.

Set C:

1. Write a C++ program to print the following pattern

```
        *
      *  *
    *  *  *
  *  *  *  *
*  *  *  *  *
```

2. Write a C++ program to calculate following series:
   1/1! + 2/2! + 3/3! + …. + n/n!

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: WellDone [ ]

Signature of Instructor

Assignment No. 2: Operators and Functions in C++

Operators in C++:

C++ has a rich set of operators. All C operators are valid in C++ also. In addition to that C++ introduces some new operators. We have already seen two such operators namely, the insertion operator <<, and the extraction operator >>. Other new operators are:

Operator	Name of Operator	Function
::	Scope resolution operator	To access global version of a variable
::*	Pointer-to-member declarator	To declare a pointer to a member of a class
->*	Pointer-to-member operator	To access a member using a pointer to the object and a pointer to that member.
.*	Pointer-to-member operator	To access a member using object name and a pointer to that member.
new	Memory allocation operator	To allocate memory for object.
delete	Memory release operator	To free allocated memory of an object.
endl	Line feed operator	To insert a new line character similar to '\n'
setw	Field width operator	To sets the field width to be used on output operations.

Scope resolution operator:

In C++ the scope resolution operator (::) is used for several reasons, some of them are:

- Accessing a global variable when there is a local variable with same name
- Defining a function outside a class
- Accessing a class's static variables
- Referring to a class inside another class
- In case of multiple Inheritance

Out of these, in this assignment we are going study first use of scope resolution operator. If the global variable name is same as that of local variable name, the scope resolution operator will be used to access the global variable.

Example: C++ program to illustrate use of Scope Resolution Operator (::)

```
#include<iostream.h>
int num = 30;                               // Initializing a global variable num
int main()
{       int num = 10;                       // Initializing the local variable num
        cout<< "\nValue of global num is " << ::num;
        cout<< "\nValue of local num is " <<num;
        return 0;
}
```

Output:

```
Value of global num is 30
Value of local num is 10
```

In above program, we have two variables both named num with global & local scope. So, to access global num variable in main function we need to use scope resolution operator (i.e. ::num).

Memory management operators:

Allocating memory of a variable or an array run time is known as Dynamic Memory Allocation(DMA).In C, dynamic memory management is handled by malloc( ) and free( ) function, but in C++ dynamic memory management is handled by using operators called 'new' and 'delete', where 'new' operator replaces malloc( ) and 'delete' operator replaces free( ) in C. New and Delete operators manage memory effectively hence they are called as memory management operators. In C++, we need to deallocate the dynamically allocated memory manually after we have no use for the variable.

Syntax for any data type:	pointer-variable = new data-type;
	delete pointer-variable;
Syntax for an array:	pointer-variable=new data-type[size];
	delete[size] pointer-variable;

Example: C++ program to illustrate use of memory management operators.

```
#include<iostream.h>
int main()
{       int* ptr;                // declare an integer pointer
        ptr = new int;           // dynamically allocate memory for an int variable
        int *arr_ptr=new int[5];         //  create a memory for an array of 5 integers
        *ptr = 100;              // assign value to the memory
        cout<< *ptr<<endl;       // print the value stored in memory
        inti;           //variable declaration anywhere in the scope is allowed in C++
        for(i=0;i<5;i++)
        {       arr_ptr[i]=i+1; //assign value to an array elements
        }
        cout<<arr_ptr[0];        // print first element of an array
        delete ptr;              // deallocate the memory
        delete[ ] arr_ptr;       //deallocate memory of an array
        return 0;
}
```

Output:

```
100
1
```

In above C++ program we have used pointer to allocate memory dynamically because new operator returns the address of memory location. In case of an array, the new operator returns the address of the first element of an array. Delete operator returns the memory to the operating system which is known as memory deallocation.

Advantages of the new operator over malloc() function:

- It does not use the sizeof() operator as it automatically computes the size of the data object.
- It automatically returns the correct data type pointer, so does not need to use the typecasting.
- It allows to initialize the data object while creating the memory space for object.

Manipulators:
Manipulators are operators that are used to change formatting parameters on streams and to insert or extract certain special characters, these are helping functions that can modify the input/output stream. It does not mean that we change the value of a variable, it only modifies the I/O stream using insertion (<<) and extraction (>>) operators. To use manipulators in C++ program we need to include header file iomanip.h.

Following are some of the most widely used C++ manipulators:

1) **endl:**
   endl is the line feed operator in C++. It acts as a stream manipulator whose purpose is to feed the whole line and then point the cursor to the beginning of the next line. We can use endl instead of '\n' (newline character) for the same purpose.
   Example:
   ```
   cout<<"Good"<<endl<<"Morning";
   ```
   This will display "Good" and "Morning" on two separate lines.
2) **setw:**
   setw manipulator function stands for set width. This manipulator is used to specify the minimum number of character positions on the output field a variable will consume, that is it sets the minimum field width on output. It is mostly used in output to right justify numbers.
   Example:
   ```
   Sum=123;
   cout<<setw(5) <<Sum;
   ```
   This sum value is right justified within the field.

		1	2	3

3) **setfill:**
   setfill is used after setw manipulator. If a value does not entirely fill a field, then the character specified in the setfill argument of the manipulator is used for filling the fields. It specifies a character that is used to fill the unused portion of a field.
   Example:
   ```
   cout<<setw(10)<<setfill('*')<<1234;
   ```
   This will give you output: ******1234
4) **setprecision:**
   The setprecision manipulator is used with floating point numbers. It is used to specify the number of digits to be displayed after the decimal point of a float value.
   Example:
   ```
   PI=3.14159;
   cout<<setprecision(2)<<PI;
   ```
   Output: 3.14

Functions in C++:

Function prototyping:
The function prototype describes the function interface and it is used to give details to the compiler about the number of arguments and about the required data types of a function parameter, it also tells about the return type of the function. Using these details, the compiler cross-checks the function signatures before calling it. If the function prototypes are not

mentioned, then the program may be compiled with some warnings. If some function is called somewhere in a program, but its body is not defined yet, that is defined after the current line, then it may generate problems. The compiler does not find what is the function and what is its signature. In that case, we need to use function prototyping. If the function is defined before, then we do not need to use prototypes.

Syntax:

```
        return_type function_name (argument_list);
```

Example:

```
        int multiplication (int x, int y, int z);
        int addition (int, int, int); /*this is also acceptable at the place of declaration because at this
stage, the compiler only checks for the type of arguments when the function is called. */
```

Call by reference:

Call by value means pass arguments by value to the function and call by reference means pass address of arguments to the function. In call by value, called function creates a new set of variable and copies the values of arguments into them. The function does not have access to the actual variables in the calling program. This mechanism is fine if the function does not want to alter the values of the original variables in calling program.

To change values of the original variables in calling program we have to use call by reference. In call by reference, address of the value is passed to the function, so actual and formal arguments share the same address space. Hence, value changed by called function will get reflected in calling function also.

Example: C++ program to illustrate use of call by reference.

```
void swap(int*, int*);
int main()
{       int a = 10, b = 20;             // initialize variables

        cout<< "Before swapping" <<endl;
        cout<< "a = " << a <<endl;
        cout<< "b = " << b <<endl;

        swap(&a, &b);                   // call function by passing variable addresses

        cout<< "\nAfter swapping" <<endl;
        cout<< "a = " << a <<endl;
        cout<< "b = " << b <<endl;
        return 0;
}

// function definition to swap numbers
void swap(int* num1, int* num2)
{       int t;
        t = *num1;
        *num1 = *num2;
        *num2 = t;
}
```

Output:

```
Before swapping
a = 10
b = 20
After swapping
a = 20
b = 10
```

In above program we are using call by reference, when the function is working with reference or address it is actually working with original data.

Return by reference:
A function can also return a reference. A C++ program can be made easier to read and maintain by using references rather than pointers. When a function returns a reference, it returns an implicit pointer to its return value. This way, a function can be used on the left side of an assignment statement.

Example: C++ program to illustrate use of return by reference.

```
#include<iostream.h>
int n;
int& test();

int main()
{
        test()=10;
        cout<<n;
        return 0;
}
int& test()
{
        return n;
}
```

In above program return type of function test() is int& hence test() returns by reference. In program, test() will not return value of n, instead it returns reference of the variable n. Since test() is returning address of n it can be assigned a value, in our program it is 10. Hence program will display output: 10.

Inline Function:
When the program executes the function call instruction, the CPU stores the memory address of the instruction following the function call, copies the arguments of the function on the stack and finally transfers control to the specified function. The CPU then executes the function code, stores the function return value in a predefined memory location/register and returns control to the calling function.
This can become overhead if the execution time of function is less than the switching time from the caller function to called function (callee). For functions that are large and/or perform complex tasks, the overhead of the function call is usually insignificant compared to the amount of time the function takes to run. However, for small, commonly-used functions, the time needed to make the

function call is often a lot more than the time needed to actually execute the function's code. This overhead occurs for small functions because execution time of small function is less than the switching time.
C++ provides an inline function feature to reduce the function call overhead. It also save overhead of arguments push/pop on the stack, while function calling. Inline function is a function that is expanded in line when it is called. If a function is Inline, the compiler places a copy of the code of that function at each point where the function is called at compile time and may make the program execution faster.
To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier, in case defined function is more than a line.

Syntax:

```
inline return-type function-name(argument list)
{
        //Function Body
}
```

Example: C++ program to illustrate use of inline function.

```
#include <iostream.h>
inline int square(int x)
{
        return (x*x);
}
int main()
{
        cout<< "Square (2): " <<square(2) <<endl;
        cout<< "Square (3): " <<square(3)<<endl;
        return 0;
}
```

Output:

```
Square (2): 4
Square (3): 9
```

Default Arguments:

In C++ programming, we can provide default values for function parameters. A default argument is a value provided in a function declaration for function parameters. If a function with default arguments is called without passing arguments, then the default values are automatically assigned by the compiler during compilation of program. However, if arguments are passed while calling the function, the default arguments are ignored.

Example: C++ program to illustrate use of default arguments.

```
#include<iostream.h>
int sum(int a=10, int b=20);
int sum(int a, int b)
{
        return (a+b);
}
```

```
int main()
{
        cout<<sum()<<endl;
        cout<<sum(50)<<endl;
        cout<<sum(50,50)<<endl;
        return 0;
}
```

Output:

```
30
70
100
```

In above program for first function call a=10 and b=20, for second function call a=50 and b=20 and for third function call a=50 and b=50.

Rules for default argument:

- A default argument is checked for type at the time of declaration and evaluated at the time of call.
- Only trailing arguments can be default values and therefore add defaults from right to left.
- We cannot provide default value to a particular argument in the middle of an argument list.

Function declaration with default values:

```
        int sum(int a, int b=20, int c=30);          //allowed
        int sum(int a=10, int b);                    //not allowed
        int sum(int a=10, int b, int c=30);          //not allowed
        int sum(int a=10, int b=20, int c=30);       //allowed
```

Default arguments are useful in situation where some arguments always have the same value. For example bank interest may remain same for all customers for a particular period of deposit.

Practice Programs:

1. Write a C++ program to read two float numbers. Perform arithmetic operations like +, - , *, / on these numbers using Inline Function. (Use manipulators)
2. Write a C++ program to store percentage of ‘n’ students and display it where ‘n’ is the number of students entered by the user.(Use new and delete operator)
3. Write a C++ program to perform increment and decrement operation on integer number. (Use inline function)

Set A:

1. Write a C++ program to accept length and width of a rectangle. Calculate and display perimeter as well as area of a rectangle by using Inline function.
2. Write a C++ program to define power function to calculate x^y. (Use default value as 2 for y).
3. Write a C++ program to accept and display Bank_Account details as Acc_No, Acc_holder_name, Addr, Contact_Number and Balance. Perform deposit of some amount and display modified bank account details. (Use manipulators)

Set B:

1. Write a C++ program to accept ‘n’ float numbers, store them in an array and print the alternate elements of an array. (Use dynamic memory allocation)
2. Write a C++ program to modify contents of an integer array. (Use Call by reference)
3. Write a C++ program to calculate area and circumference of a Circle. (Use default argument, scope resolution operator and manipulator.)

Set C:

1. Create a C++ program to maintain inventory of a book having details Title, Authors[], Price, Publisher and Stock. Book can be sold, if stock is available, otherwise purchase will be made. Write a menu driven program to perform following operation:
   - Accept book details.
   - Sale a book. (Sale contains number of copies to be sold.)
   - Purchase a book. (Purchase contains number of copies to be purchased)

   (Use dynamic memory allocation while accepting author details).

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 3: Classes and Objects

Class:

A class in C++ is just an extension of a 'structure' used in the 'C' language. Class is a user-defined data type. It actually binds the data and its related functions in one unit, they are called members of the class.

A structure and a class differ a lot as a structure has limited functionality and features as compared to a class. A structure is used to represent a record and a class can have both data members and functions also. C++ expands the role of structure to create a class.

The **Structure and Class**, are almost similar in all respect except the significant one difference that, structure by default have all its member as "public", and class by default have all its member "private". Both a structure and a class provide a way to create a customized data type which can be used further to create instances. Instance of structure is called 'structure variable' and instance of a class is called 'object'.

Object:

An object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Access Specifiers:

Access specifiers are used to implement an important feature of Object-Oriented Programming known as Data hiding. Access specifiers in a class define how the data members and functions of a class can be accessed. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions. This access restriction to the class members is specified by the labeled public, private, and protected sections within the class body. The keywords public, private, and protected are called access specifiers.

- public - members are accessible from outside the class but within a program.
- private - members cannot be accessed or viewed from outside the class. Only the class and friend functions can access private members.
- protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

But if we do not specify any access specifier for the members inside the class then by default the access specifier for the members will be private. Member functions of the class can access all the data members and other member functions of the same class (private, public or protected) directly by using their names.

Example: C++ program to demonstrate class, object, access specifiers and defining member function inside class definition.

```
#include<iostream.h>
class Square                        //class
{
        public:                     //access specifier
        float side;
        float area()                //member function definition inside the class
        {       return(side*side);
        }
};
int main()                                  // main function
{
        Square obj;                         //object
        obj.side = 5.5;             // accessing public data member outside class
        cout<< "Square side length is: " <<obj.side<< "\n";
        cout<< "Area of square is: " <<obj.area();
        return 0;
}
```

Output:

```
Square side length is: 5.5
Area of square is: 30.25
```

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed by a semicolon or a list of declarations.

In C++ **public keyword** determines the access attributes of the members of the class that follows it, in above program data member side and member function area are public.

A public member can be accessed from outside the class anywhere within the scope of the class object hence side is accessible in main function through object of square class. You can also specify the members of a class as **private** or **protected** as per the need.

Defining member functions inside and outside class definition:

Member functions are the functions, which have their declaration inside the class definition and works on the data members of the class. The definition of member functions can be inside or outside the definition of class. In both the cases, the function body remains the same; however, the function header is different.

Member function definition inside the class definition:

If the member function is defined inside the class definition it can be defined directly. Member function inside the class does not require to be declared first here we can directly define the function. Defining a member function within the class definition declares the function by default **inline**, even if you do not use the inline specifier. Above C++ program is an example of member function definition inside the class.

Member function definition outside the class definition:
If the member function is defined outside the class, then we have to use the scope resolution operator '::' along with class name and function name. Function name in the function header is preceded by the class name and the scope resolution operator (: :).
The scope resolution operator informs the compiler what class the member belongs to. Defining a member function outside a class requires the function declaration (function prototype) to be provided inside the class definition.
Example:
```
#include<iostream.h>
class Square
{
        public:
                float side;
                float area();
};
float Square::area()                    //member function definition outside the class
{       return (side*side);
 }
int main()
{
        Square obj;
        obj.side=5.5;
        cout<< "Square side length is: " <<obj.side<< "\n";
        cout<< "Area of square is: " <<obj.area();
        return 0;
}
```
Output:
```
Square side length is: 5.5
Area of square is: 30.25
```

Static data members and Static member functions:

Static data members:
Static data members are class members that are declared using the static keyword. The normal variable is created when the function is called and its scope is limited, while the static variable is created once and destroyed at the end of the program. These variables are visible within the class but its lifetime is till the program ends. There is only one copy of the static data member in the class, even if there are many class objects. This is because all the objects share the static data member. To hold the count of objects created for a class, static data members are used.
The static data member is always initialized to zero when the first class object is created. While defining a static variable, some initial value can also be initialized to the variable. Type and scope of each static member variable must be defined outside the class definition using scope resolution operator. This is necessary because the static data members are stored separately rather than as a part of an object.
Static data members are associated with the class itself rather than with any class object, hence they are also known as class variables.

Static member functions:

Like static data member, we can also have static member functions. A static member function can only access other static variables or functions present in the same class. To create a static member function we need to use the static keyword while declaring the function.

Since static member variables are class properties and not object properties, to access them we need to use the class name instead of the object name. A static member function can be called even if no objects of the class exist and the static functions are accessed using class name and the scope resolution operator ::. You could use a static member function to determine whether some objects of the class have been created or not.

Example: C++ program to illustrate use of static data member and static member function.

```
#include <iostream.h>
class StaticDemo
{
        private:
                static int num;                 //declaration of static data member
        public:
                static void Display()           //static member function definition
                {
                        cout<<"Value of num is : "<<num<<endl;//accessing static data member
                }
};
int StaticDemo :: num=10;     //static data member definition and initialization outside class
int main()
{
        StaticDemo::Display();                  //call to static member function
        return 0;
}
```

Output:

```
Value of num is : 10
```

Array of objects:

An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of object. An array which contains the class type of element is called array of objects.

Array of objects contains the objects of the class as its individual elements. It is declared in the same way as an array of any built-in data type.

Example: C++ program to illustrate use of array of objects.

```
#include<iostream.h>
class Employee
{
        int Emp_id;
        char Name[20];
        long Salary;
```

```
	public:
	void Accept()
	{
		cout<<"\n\tEnter Employee Id, Name and Salary : ";
		cin>>Emp_id>>Name>>Salary;
	}
	void Display()
	{
		cout<<"\n"<<Emp_id<<"\t"<<Name<<"\t"<<Salary;
	}
};
int main()
{
	int i;
	Employee emp[3];		//Creating Array of objects to store 3 Employees details
	for(i=0;i<3;i++)
	{
		cout<<"\nEnter details of "<<i+1<<" Employee";
		emp[i].Accept();
	}
	cout<<"\nDetails of Employees";
	for(i=0;i<3;i++)
	emp[i].Display();
	return 0;
}
```

Above program will accept and display details of 3 employees using array of objects.

Objects as a function argument:
In C++ we can pass objects of a class as arguments, the same way how we pass other variables. To pass it we write the object name as the argument while calling the function. Object as function argument is normally used to communicate between two objects.
The objects of a class can be passed as arguments to member functions as well as non-member functions either by value or by reference.
Call by value: When an object is passed by value, a copy of the actual object is created inside the function, to pass entire object into another function. This copy is destroyed when the function terminates. Moreover, any changes made to the copy of the object inside the function are not reflected in the actual object.
Call by reference: In this method, only a reference to that object (not the entire object) is passed to the function. Thus, the changes made to the object within the function are also reflected in the actual object.

Whenever an object of a class is passed to a member function of the same class, its data members can be accessed inside the function using the object name and the dot operator. However, the data members of the calling object can be directly accessed inside the function without using the object name and the dot operator.

Function returning objects:
As we can pass entire object as an argument, similarly we can return object from the function. We can return entire object from function by specifying its return type as class name just like primary data-types. An object can be returned by a function using the return keyword.

Friend Function:
Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class. Similarly, protected members can only be accessed by derived classes and are inaccessible from outside. However, there is a feature in C++ called **friend functions** that break this rule and allow us to access **private** and **protected** data of a class outside the class.
For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend. A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class.
Even though the prototypes for friend functions appear in the class definition, friends are not member functions. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword friend or scope resolution operator.
Characteristics of a Friend Function:

- Friend function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

Example: C++ program to illustrate use of objects as a function argument, function returning object and friend function.
```
# include <iostream.h>
class Demo
{
        int x,y;
        public:
        void Accept();
        // friend function declaration with objects as arguments and returning object
        friend Demo sum (Demo, Demo);
        void Display();
};
Demo sum(Demo obj1, Demo obj2)
{
        Demo obj3;
        obj3.x=obj1.x+obj2.x;
        obj3.y=obj1.y+obj2.y;
        return obj3;                //function returning object
}
```

```
int main()
{
        Demo obj1, obj2, obj3;
        obj1.Accept();
        obj2.Accept();
        obj3=sum(obj1,obj2);        //call to a friend function
        obj3.Display();
        return 0;
}

void Demo::Accept()
{
        cout<<"\nPlease enter value of x and y :";
        cin>>x>>y;
}

void Demo::Display()
{
        cout<<"x= "<<x<<endl;
        cout<<"y= "<<y<<endl;
}
```

Output:

```
Please enter value of x and y : 10 20
Please enter value of x and y : 10 20
x= 20
y= 40
```

Above C++ example give us an idea about the concept of a friend function, but it doesn't show any meaningful use. In the above example, we could have made "sum" as a member function of the class instead of declaring it as a friend function to the class.
A more meaningful use would be operating on objects of two different classes. That's when the friend function can be very helpful. A friend function can act as a bridge between two classes as in the following example.

Example: C++ program to illustrate use of friend function for two classes.

```
#include <iostream.h>

class Square;  // forward declaration of a class

class Rectangle
{
      int width, height;
      public:
      void setvalue(int w, int h){width=w; height=h;}
      friend void display(Rectangle &, Square &);
};
```

```
class Square
{
	int side;
	public:
	void setvalue(int s){side=s;}
	friend void display(Rectangle &, Square &);
};

void display(Rectangle &r, Square &s)
{
	cout<< "Rectangle Area: " <<r.width * r.height<<endl;
	cout<< "Square Area: " <<s.side * s.side<<endl;
}
int main ()
{
	Rectangle rec;
	rec.setvalue(5,10);
	Square sq;
	sq.setvalue(5);
	display(rec,sq);
	return 0;
}
```
Output:
```
Rectangle Area: 50
Square Area: 25
```
In above program friend function display() is friendly to Rectangle and Square class. It does not belong to any class, so it can be used to access private data of Rectangle and Square class.

Friend Class:
Like friend function, a class can also be a friend of another class. A friend class can access all the private and protected members of other class in which it is declared as friend. This is needed when we want to allow a particular class to access the private and protected members of a class. In order to access the private and protected members of a class into friend class we must pass on object of a class to the member functions of friend class.

Example: C++ program to illustrate use of friend class.
```
#include <iostream.h>
class A
{
	int num;
	public:
	void setvalue(int i)
	{
		num=i;
	}
```

```
        friend class B;         //making B class, a friend class of A class

};
class B
{
        public:
        void display(A &a)
        {
                cout<<"Value of num is : "<<a.num;
        }
};
int main()
{
        A a_obj;
        a_obj.setvalue(10);
        B b_obj;
        b_obj.display(a_obj);
        return 0;
}
```

Output:

```
Value of x is :10
```

In the above example, B class is a friend class of A class. In order to access the private members of A class into B class we have explicitly pass an object of A class to the member functions of B class.
This is similar to passing an object as function argument but the difference is, an object a_obj we are passing as argument is of different class (A) and the calling object is of different class (B).

Practice Programs:

1. Write a C++ program to create a class Customer with data members ID, Name, Addr and Contact_No. Write member functions to accept and display customer information. (Use scope resolution operator while defining member functions)
2. Write a C++ program to create a class Employee with data members Emp_id, Name, department, date_of_joining and Salary. Write member functions to accept and display details of 'n' employees. (Use array of objects)
3. Write a C++ program to add two float numbers of two different classes using friend function.

Set A:

1. Write a C++ program to create a class Student with data members Roll_No, Student_Name, Class. Write member functions to accept and display Student information also display count of students. (Use Static data member and Static member function)
2. Write a C++ program to calculate the average height of all the students of a class. The number of students and their heights are entered by user. (Use array of objects)

3. Write a C++ program to calculate maximum and minimum of two integer numbers of two different classes.(Use friend function)

Set B:

1. Write a C++ program using class to accept and display 'n' Products information, also display information of a product having maximum price. (Use array of objects and dynamic memory allocation)
2. Write a C++ program to create a class Distance with data members feet and inches. Write member functions for the following:
    a. To accept distance
    b. To display distance
    c. To add two distance objects
    (Use object as a function argument and function returning object)
3. Write a C++ program to create two classes Array1 and Array2 with an integer array as a data member. Write necessary member functions to accept and display array elements of both the classes. Find and display maximum of both the array. (Use Friend function)

Set C:

1. Write a C++ program to calculate multiplication of two integer numbers of two different classes. (Use friend class)

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 4: Constructors and Destructors

Constructor:
A constructor is a 'special' member function whose task is to initialize the objects of its class. It is called constructor because it constructs the values of data members of the class. Constructor is automatically called when object of class is created.

Characteristics of Constructor:
- Constructors are declared as public member function.
- Constructors are automatically invoked when an object of class is created.
- Constructor has same name as the class name.
- Constructors don't have any return type.
- Constructors can have default arguments.
- Constructors cannot be inherited, though a derived class can call the base class constructor.
- Constructors cannot be virtual.
- Constructors cannot refer to their addresses.
- Constructors can implicitly call new and delete operators when memory allocation is required.

Constructors can be defined either inside the class definition or outside class definition. If constructors are defined outside class definition, then they can be defined using class name and scope resolution:: operator.

Example: To define Constructor inside the class.
```
class Number
{
        int n;
    public:
        Number( )                       //Constructor defined inside the class
        {
            n=10;
        }
};
```

Example: To define Constructor outside the class.
```
class Number
{
        int n;
    public:
        Number( );                      //Constructor declared
 };
 Number :: Number ( )                   //Constructor Defined outside the class
{
        n=10;
 }
```
When a class contains a constructor, objects of the class will be initialized automatically.

Ex. Number Obj1;

Here Obj1 invokes constructor and initializes the data members of class Number.
If constructor is not defined in a class, C++ compiler generates a default constructor.

Types of Constructors:

1. **Default Constructors:**

   The constructor that accepts **no arguments** is called as Default Constructor.

   Example: To illustrate the use of Default Constructor.

```
#include<iostream.h>
class Number
{
        int n;
   public:
        Number( )               //Default Constructor
        {
                n = 0;
        }
};
int main( )
{
        Number Obj1;
        return 0;
}
```

   Number Obj1 invokes Default constructor and initializes data member n to 0(zero).

2. **Parameterized Constructors:**

   The constructor that **accepts arguments** is called as Parameterized constructor. These arguments initialize an object, when it is created. The constructors can be called explicitly or implicitly.
   If more than one constructor is defined in a class, it is called as **Constructor Overloading.**

   Example: To illustrate the use of Parameterized Constructor.

```
#include<iostream.h>
class Number
{
        int n;
   public:
        Number(int x )          //Parameterized Constructor
        {
                n = x;
        }
};
int main( )
```

```
{
        Number Obj1 = Number(50);           // Explicit call
        Number Obj2(100);                   // Implicit call
}
```

Number Obj1 & Number Obj2 invokes parameterized constructor and initializes data member n to 50 & 100 respectively.

3. **Copy Constructor:**

   A constructor that **initializes an object** using another object of the same class is called as copy constructor. It takes a reference of object of the same class as its argument. It copies data from one object to other by copying every member of an object with the member of object passed as argument.

   Example: To illustrate the use of Copy Constructor.

```
#include<iostream.h>
class Number
{
        int n;
  public:
        Number (int x )
        {
                n = x;
        }
        Number (Number &N)          //Copy Constructor
        {
                n= N.n;
        }
};

int main( )
{
        Number Obj1(10), Obj2(Obj1);
        return 0;
}
```

   Number Obj2(Obj1) defines the obj2 and at the same time initializes it to values of Obj1.

4. **Dynamic Constructor:**

   The constructor can be used to allocate memory while creating objects. Memory can be allocated using new operator. Allocation of memory to objects at the time of their construction is known as dynamic construction of objects.

   Example: To illustrate the use of Dynamic Constructor.

```
#include<iostream.h>
#include<string.h>
```

```
class MyString
{
        Char *Str;
        int len;
  public:
        MyString ()
        {
                len=0;
                Str=new char [len+1];
        }
        MyString (char *S)
        {
                len=strlen(S);
                Str=new char [len+1];
                Strcpy(Str, S);
        }
        void Concatenate(MyString &S1, MyString &S2)
        {
                len=S1.len+S2.len;
                delete Str;
                Str=new char [len+1];
                Strcpy(Str, S1.Str);
                Strcat(Str, S2.Str);
                cout<<"String ="<<Str;
        }
};
int main( )
{
        MyString Obj1("Computer"), Obj2("Application"), Obj3;
        Obj3.Concatenate(Obj1, Obj2);
        return 0;
}
```

Constructors with default arguments

It is possible to define constructors with default arguments.

Example: To illustrate the use of Constructors with default arguments.

```
#include<iostream.h>
class Number
{
        int m, n;
   public:
        Number(int x, int y=100 )
        {
                m = x;
                n=y;
        }
};
int main( )
{
        Number Obj1(50);
}
```

Number Obj1 invokes constructor with default arguments and assigns the value 50 to the variable x and 100 to y.

Dynamic initialization of Objects:

Class objects can be initialized dynamically i.e. initial value of an object can be provided during run time. Dynamic initialization is used to provide various initialization formats, using overloaded constructors.

Example: To illustrate the use of Dynamic initialization of Constructor

```
#include<iostream.h>
class Number
{
        int n;
    public:
        Number(int x)
        {
                n = x;
        }
        void display()
        {
                cout<<"n = " <<n;
        }
};
int main( )
{
        int a;
        cout<<"\n Enter the value of a:";
        cin>>a;
        Number Obj(a);
        Obj.display();
        return 0;
}
```

Destructor:

Destructor is a member function that destroys an object which has been created by constructor. If new operator is used to allocate memory in the constructors, delete operator is used to free memory in the destructor. Destructor can clean up the storage which is no longer accessible.

A destructor is invoked implicitly when the object goes out of scope like:

a. the function ends.
b. the program ends.

c. a block containing local variables ends.
d. a delete operator is called.

Characteristics of Destructor:

- Destructors have same name as the class name preceded by a tilde (~).
- Destructors doesn’t take any argument and doesn’t return any value.

Example: To illustrate the use of Destructor.

```
#include<iostream.h>
class Number
{
    public:
        Number ( )
        {
                cout<<”\n Constructor called”;
        }
        ~Number ( )
        {
                cout<<”\n Destructor called”;
        }

};
int main( )
{
        Number Obj1;
        {
                Number Obj2;
        }                                   //Destructor Ob2 called
        return 0;                           //Destructor Ob1 called
}
```

Note: Objects are destroyed in the reverse order of creation.

Practice Programs:

1. Write a C++ program to create a class ‘MyNumber’ with three data members of type integer. Create and initialize the object using default constructor and parameterized constructor. Also define copy constructor to copy one object to another. Write a C++ program to illustrate the use of above class.

2. Write a C++ program to create a class ‘Fraction’ with integer data members numerator and denominator. Create and initialize the object using parameterized constructor. Write a member function to display addition two fraction objects.(Use the concept of dynamic initialization of object)

3. Write a C++ program to create a class ‘MyArray’ which contains single dimensional integer array of given size. Write a member function to display array in ascending order. (Use Dynamic Constructor to allocate and Destructor to free memory of an object)

Set A:

1. Write a C++ program to create a class 'MyNumber' with three data members of type integer. Create and initialize the object using default constructor, parameterized constructor and parameterized constructor with default value. Write a member function to display average of given three numbers for all objects.

2. Write a C++ program to create a class MyDate with three data members as dd, mm, yyyy. Create and initialize the object by using parameterized constructor and display date in dd-mon-yyyy format. (Input: 19-12-2014 Output: 19-Dec-2014).(Use the concept of dynamic initialization of object)

3. Write a C++ program to create a class 'MyPoint' with two integer data members as x & y. Define copy constructor to copy one object to another. (Use Default and parameterized constructor to initialize the appropriate objects) Write a C++ program to illustrate the use of above class.

Set B:

1. Write a C++ program to create a class 'MyArray' which contains single dimensional integer array of given size. Write a member function to display even and odd numbers from a given array. (Use Dynamic Constructor to allocate and Destructor to free memory of an object)

2. Write a C++ program to create a class 'MyMatrix' which contains two dimensional integer array of size mXn. Write a member function to display sum of all elements of entered matrix. (Use Dynamic Constructor for allocating memory and Destructor to free memory of an object)

3. Write a C++ program to create a class 'MyVector' with data members size & a pointer to integer. The size of the vector varies so the memory should be allocated dynamically. Create and initialize the object using default and parameterized constructor. Write a member function to display the vector in the format (10, 20, 30,….)

Set C:

1. Create a C++ class 'Student' with data members Rollno, Name, Number of subjects, Marks of each subject (Number of subjects varies for each student). Write a parameterized constructor which initializes rollno, name & Number of subjects and creates the array of marks dynamically. Display the details of all students with percentage and class obtained.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 5: Inheritance

Inheritance:
The mechanism of deriving a new class from an old class is called as **Inheritance**.
Inheritance allows a derived class to inherit the properties and characteristics from base class. A class can also inherit properties from more than one class or from more than one level. Inheritance supports the reusability as inheritance can extend the use of existing classes and eliminate redundant code.

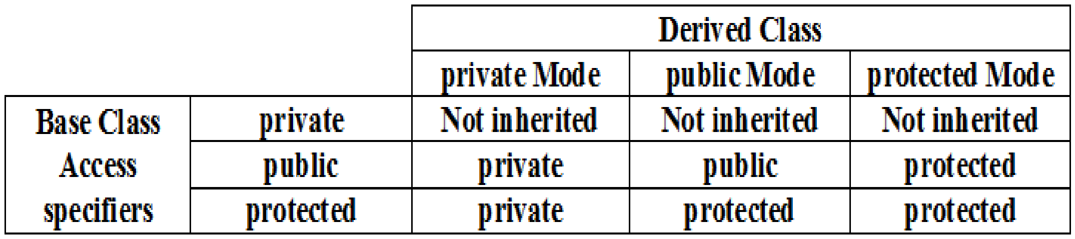
The class that inherits the properties from another class is called Sub class or **Derived Class**. The class whose properties are inherited by derived class is called Super class or **Base Class.**
Syntax to define derived class:

```
class Derived_class_name : visibility_mode Base_class_name
        {
                //body of Derived class
        };
```

Where,
Derived_class_name is the name of the sub class/derived class.
visibility_mode specifies the mode in which derived class can be inherited. For example: public, private, protected. Default visibility mode is private.
Base_class_name is the name of the base class from which you want to inherit the sub class.

Modes of Inheritance

The following table represents the scope of the access specifier of the members of base class in the derived class when derived in private, public & protected modes:



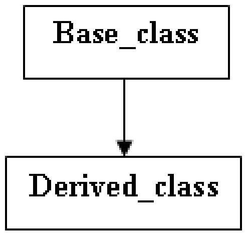
		Derived Class		
		private Mode	public Mode	protected Mode
Base Class Access specifiers	private	Not inherited	Not inherited	Not inherited
	public	private	public	protected
	protected	private	protected	protected

- **Private mode**: If a sub class is derived from a base class in private mode then both public member and protected members of the base class becomes Private in derived class. Private members of the base class never get inherited in sub class.
- **Public mode**: If a sub class is derived from base class in public mode then the public member of the base class remains public in the derived class and protected members of the base class remains protected in derived class. Private members of the base class never get inherited in sub class.
- **Protected mode**: If a sub class is derived from a base class in protected mode then both public member and protected members of the base class becomes protected in derived class. Private members of the base class never get inherited in sub class.

Types of Inheritance:

1. Single Inheritance:

A derived class with **only one base class** is called as Single Inheritance.



Syntax to define derived class:

```
class Derived_class: visibility_mode Base_class
{
        //Body of Derived class
};
```

Example: To illustrate the use of Single Inheritance using public derivation.

```
#include<iostream.h>
class Base
{
        int x;                          //private; not inheritable
    public:
        int y;                          //public; inheritable
        void setValues()
        {
                x=10;
                y=20;
        }
        int getx()
        {
                return x;
        }
}
class Derived : public Base             // public derivation
{
        int z;
   public:
        void add()
        {
                Z=getx() + y;
        }
        void display()
        {
                cout<<"\n x= "<<getx();
                cout<<"\n y= "<<y;
                cout<<"\n Addition : "<<z
        }
}
int main()
{
        Derived D;
        D.setValues();
        D.add();
        D.display();
}
```

Derived class is a public derivation of the base class Base. So, Derived class inherits all the public members of class Base and retains their visibility. Thus public members of the Base class are also public members of the Derived class. The private members of the Base class cannot be inherited by class Derived.

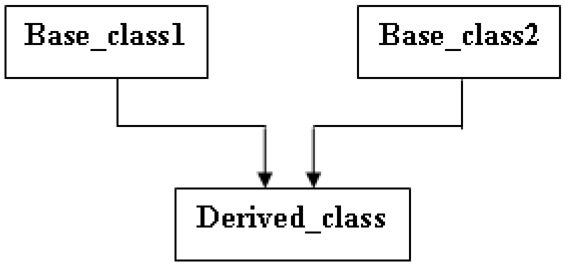
Example: To illustrate the use of Single Inheritance using private derivation.

```
#include<iostream.h>
class Base
{
        int x;                          //private; not inheritable
   public:
        int y;                          //public; ready for  inheritance
        void setValues()
        {
                x=10;
                y=20;
        }
        int getx()
        {
                return x;
        }
}
class Derived : private Base            // private derivation
{
        int z;
  public:
        void add()
        {
                D.setValues();
                z=getx() + y;
        }
        void display()
        {
                cout<<"\n x= "<<getx();
                cout<<"\n y= "<<y;
                cout<<"\n Addition : "<<z
        }
}
int main()
{
        Derived D;
        // D.setValues();                    wont work
        D.add()
        D.display();
}
```

Derived class is a private derivation of the base class Base. So, Derived class inherits only public members of base class Base as private and retains their visibility. The private members of the Base class cannot be inherited by class Derived.

2. Multiple Inheritance:

A derived class with **several base classes** is called as Multiple Inheritance.



Syntax to define derived class:

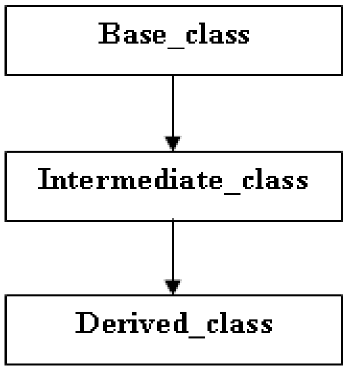
class **Derived_class** : **visibility_mode** Base_class1, **visibility_mode** Base_class2,..

```
{
        //Body of Derived class
};
```

A class is derived with multiple base classes. The number of base classes are separated by a comma (‘, ‘). Visibility mode for every base class must be specified.

3. Multilevel Inheritance:

The mechanism of **deriving a class from another derived class** is called as Multilevel inheritance.

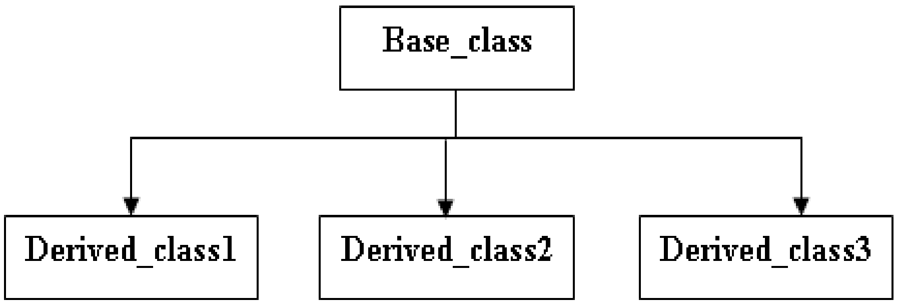


Syntax to define derived class:

```
class Intermediate_class : visibility_mode Base_ class
{
        //Body of Intermediate class
};
class Derived_class : visibility_mode Intermediate_class
{
        //Body of Derived class
};
```

4. **Hierarchical Inheritance**:

More than one derived classes inherits the features from a **single base class** is called as Hierarchical Inheritance i.e. more than one derived classes are created from a single base class.



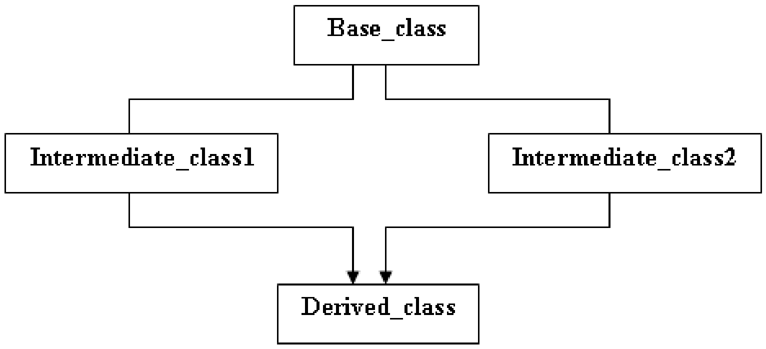
Syntax to define derived class:

```
class Derived_class1 : visibility_mode Base_class
{
        //Body of Derived class
};
class Derived_class2 : visibility_mode Base_class
{
        //Body of Derived class
};
class Derived_class3 : visibility_mode Base_class
{
        //Body of Derived class
};
```

5. **Hybrid Inheritance**:

More than one type of inheritance is combined to form Hybrid Inheritance.

For Ex.: Combination of Hierarchical inheritance and Multiple Inheritance.



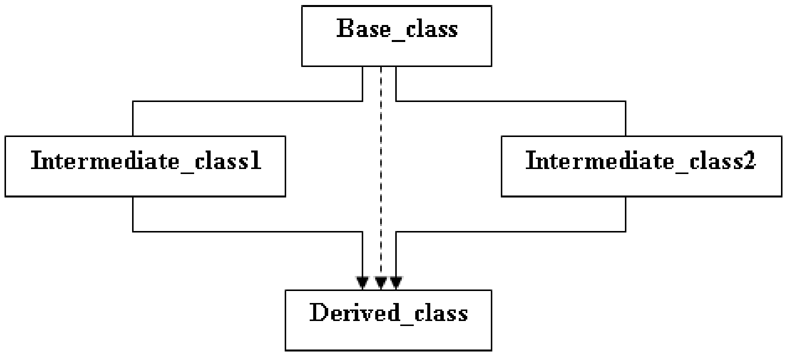
Syntax to define derived class:
class **Intermediate_class1** : **visibility_mode** Base_ class
  {
        //Body of Intermediate class1
};
class **Intermediate_class2** : **visibility_mode** Base_ class
{
        //Body of Intermediate class2
};
class **Derived_class** : **visibility_mode** Intermediate_class1, **visibility_mode** Intermediate_class2
{
        //Body of Derived class
};

Virtual base class:

Several paths exist to a derived class from the same base class i.e. a derived class can have duplicate sets of members inherited from a single base class. This introduces **ambiguity and it should be avoided.**

Duplication of inherited members due to multiple paths is avoided by making the common base class as **virtual base class**. This is achieved by preceding the base class name with the keyword **virtual**.

When a class is made a virtual base class, necessary care is taken so that only one copy of that class is inherited, regardless of the number of paths exist between virtual base class and a derived class.



Syntax:

```
class Base_class
{
        //Body of Base class
 };
class Intermediate_class1 : virtual visibility_mode Base_ class
{
         //Body of Intermediate class1
};
class Intermediate_class2 : visibility_mode virtual Base_ class
{
         //Body of Intermediate class2
};
class Derived_class : visibility_mode Intermediate_class1, visibility_mode Intermediate_class2
{
         //Body of Derived class
};
```

Note: Virtual and visibility mode can be used in either order.

Example: To illustrate the use of virtual base class.
Calculating marks and grade of student using virtual base class.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
int n;
class student
{
  protected:
        int rno;
        char name[20];
  public:
        void acceptinfo()
        {
                cout<<"\nRoll no: ";
                cin>>rno;
                cout<<"Name: ";
                gets(name);
        }
        void displayinfo()
        {
                cout<<"\nRoll no: "<<rno <<"\nName: "<<name;
        }
};
```

```
class test:public virtual student
{
  protected:
           int marks1, marks2;
  public:
           void acceptmark()
           {
                     cout<<"Mark 1: ";
                     cin>>marks1;
                     cout<<"Mark 2: ";
                     cin>>marks2;
           }
           void displaymark()
           {
                     cout<<"\nMark 1: "<<marks1<<"\nMark 2: "<<marks2;
           }
};
class sport : public virtual student
{
  protected:
           int score;
  public:
           void acceptscore()
           {
                     cout<<"Score: ";
                     cin>>score;
           }
           void displayscore()
           {
                     cout<<"\nScore: "<<score;
           }
};
class result:public test,public sport
{
  protected:
           int totalmarks, perc;
           char grade[20];
  public:
           void calctotal();
           void accept()
           {
                     acceptinfo();
                     acceptmark();
                     acceptscore();
                     calctotal();
           }
```

```
        void display()
         {
                displayinfo();
                displaymark();
                displayscore();
                cout<<"\nTotal: "<<totalmarks
                        <<"\nPercentage: "<<perc<<" %"
                        <<"\nGrade: "<<grade<<"\n";
         }
        int gettotal()
         {
                return totalmarks;
         }
};
void result::calctotal()
{
        totalmarks=marks1+marks2+score;
        perc=(totalmarks*100)/300;
        if(perc>=75)
                strcpy(grade,"Distinction");
        else if(perc>=60 && perc <75)
                strcpy(grade,"First Class");
        else if(perc>=35 && perc<60)
                strcpy(grade,"Pass Class");
        else
                strcpy(grade,"Fail");
}

int main()
{
        int ch, i, j;
        clrscr();
        do{

                cout<<"\nMain Menu\n"
                        <<"\n1. Accept details"
                        <<"\n2. Display details in ascending order"
                        <<"\n3. Exit\n"
                        <<"\nEnter your option: ";
                cin>>ch;
                switch(ch)
                {
                        case 1: cout<<"\nDetails of how many students do you want ot enter: ";
                                cin>>n;
                                result r[10];
                                cout<<"\nEnter the following details";
```

```
                    for(i=0; i<n; i++)
                    {
                            r[i].accept();
                    }
                    break;
            case 2: cout<<"\nThe details are\n";
                    for(i=0; i<n; i++)
                    {
                            r[i].display();
                    }
                    getch();
                    break;
            case 3: exit(0);
        }
    }while(ch!=3);
    return 0;
}
```

Abstract class:

An abstract class is **not used to create objects**. An abstract class is designed only to acts as a base class.

Constructor in derived class:

While using constructors during inheritance, is that, as long as a base class constructor does not take any arguments, the derived class need not have a constructor function. However, if a base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor. While applying inheritance, we usually create objects using derived class. Thus, it makes sense for the derived class to pass arguments to the base class constructor. When both the derived and base class contains constructors, the base constructor is executed first and then the constructor in the derived class is executed.

In case of multiple inheritance, the base class is constructed in the same order in which they appear in the declaration of the derived class. Similarly, in a multilevel inheritance, the constructor will be executed in the order of inheritance.

The derived class takes the responsibility of supplying the initial values to its base class. The constructor of the derived class receives the entire list of required values as its argument and passes them on to the base constructor in the order in which they are declared in the derived class. A base class constructor is called and executed before executing the statements in the body of the derived class.

Syntax to define derived class constructor:

```
Derived-Constructor (ArgList2, ArgList2,…….ArgListN,ArgListD): Base1(ArgList1),
Base2(ArgList2)……. BaseN(ArgListN)
{
    // Body of Derived Constructor
}
```

Practice Programs:

1. Create a base class Employee(empcode, empname). Derive the classes Manager(designation, club_dues), Scientist(deptname, publications) and Labourer from Employee class. Write a C++ menu driven program
   i. to accept the details of 'n' employees
   ii. to display the information
   iii. to display details of manager with designation as "General Manger".

2. Create two base classes Learning_Info(Roll_No, Stud_Name, Class, Percentage) and Earning_Info(No_of_hours_worked, Charges_per_hour). Derive a class Earn_Learn_info from above two classes. Write necessary member functions to accept and display Student information. Calculate total money earned by the student. **(Use constructor in derived class)**

Set A:

1. Design a base class Product(Product_Id, Product_Name, Price). Derive a class Discount (Discount_In_Percentage) from Product. A customer buys 'n' products. Write a C++ program to calculate total price, total discount.

2. Design a Base class Customer (name, phone-number).Derive a class Depositor(accno, balance) from Customer. Again derive a class Borrower (loan-no, loan-amt) from Depositor. Write necessary member functions to read and display the details of 'n' customers.

Set B:

1. Design two base classes Personnel (name, address, email-id, birth date) and Academic (marks in tenth, marks in twelth, class obtained). Derive a class Bio-data from both these classes. Write a C++ program to prepare a bio-data of a student having Personnel and Academic information.

2. Design a base class Employee (emp-code, name). Derive two classes as Fulltime (daily rate, number of days, salary) and Parttime (number of working hours, hourly rate, salary) from Employee. Write a C++ menu driven program to:
   i. Accept the details of 'n' employees and calculate the salary.
   ii. Display the details of 'n' employees.
   iii. Search a given Employee.

3. Create a base class Student(Roll_No, Name) which derives two classes Academic_Marks(Mark1, Mark2, Mark3) and Extra_Activities_Marks(Marks). Class Result(Total_Marks, Grade) inherits both Academic_Marks and Extra_Activities_Marks classes. (Use Virtual Base Class)
   Write a C++ menu driven program to perform the following functions:
   i. Build a master table.
   ii. Calculate Total_marks and grade.

Set C:

1. Create a base class Student(Roll_No, Name, Class) which derives two classes Internal_Marks(IntM1, IntM2, IntM3, IntM4, IntM5) and External_Marks(ExtM1 ExtM2, ExtM3, ExtM4, ExtM5). Class Result(T1, T2, T3, T4, T5) inherits both Internal_Marks and External_Marks classes. (Use Virtual Base Class)
   Write a C++ menu driven program to perform the following functions:
   i. To Accept and display student details
   ii. Calculate Subject wise total marks obtained.
   iii. Check whether student has passed in Internal and External Exam of each subject. Also check whether he has passed in respective subject or not and display result accordingly.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 6: Polymorphism

Polymorphism:
Polymorphism means ‘One name, multiple forms’.

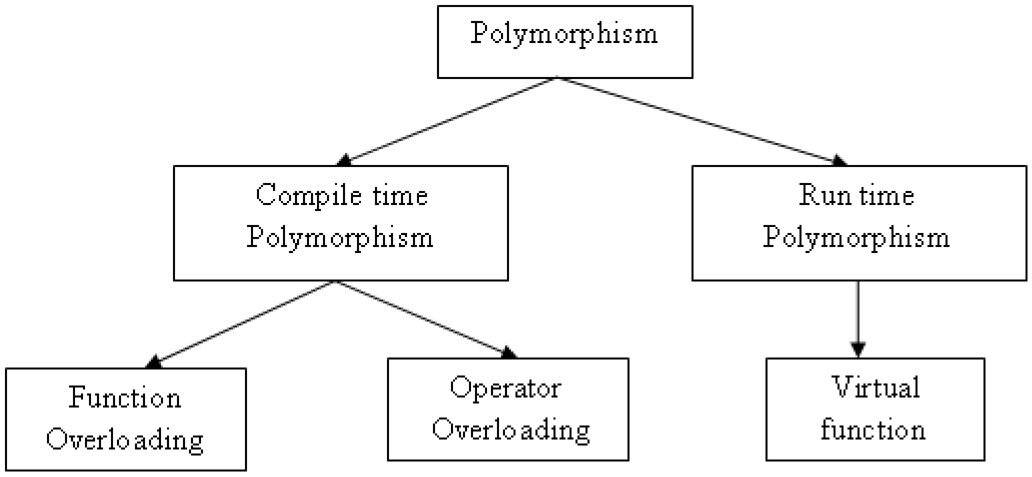


Fig.: Achieving Polymorphism

Compile time Polymorphism:

Compiler selects appropriate function for a particular call at compile time is called as **Compile time Polymorphism.** It is also called as early binding or static binding or static linking. Compile time Polymorphism is achieved by using function overloading and operator overloading.

Function Overloading:

Same function name is used to create a function that perform variety of different tasks is called as Function Overloading.

A family of functions can be designed with same function name but with different argument list. The function performs different operations depending on the argument list in the function call. The appropriate function to be invoked is determined by checking the number and type of arguments.

Example: C++ **program to find volume of cube, cylinder and rectangle using function overloading.**

```
#include<iostream.h>
#include<conio.h>
int volume(int);
double volume(double,int);
long volume(long,int,int);
int main()
{
        clrscr();
        int x,s,h,b;
```

```
	double y,r;
	long z,l;
	cout<<"\nEnter the value for s: ";
	cin>>s;
	x=volume(s);
	cout<<"\nVolume of cube: "<<x;

	cout<<"\nEnter the value for r and h: ";
	cin>>r>>h;
	y=volume(r,h);
	cout<<"\nVolume of cylinder: "<<y;

	cout<<"\nEnter the value for l,b and h: ";
	cin>>l>>b>>h;
	z=volume(l,b,h);
	cout<<"\nVolume of rectangle: "<<z;
	getch();
	return 0;
}
int volume(int s)
{
	return (s*s*s);
}
double volume(double r,int h)
{
	return (3.14*r*r*h);
}
long volume(long l,int b,int h)
{
	return (l*b*h);
}
```

Example: C++ program to find area of triangle, circle, and rectangle using function overloading.

```
#include<iostream.h>
#include<conio.h>
float area(float r)
{
	return(3.14*r*r);
}

float area(float b,int h)
{
	return(0.5*b*h);
}
```

```
int area(int l,int b)
{
        return(l*b);
}
void disp(float m)
{
        cout<<"\nArea: "<<m;
}
int main()
{
        clrscr();
        int m,n;
        float l,a;
        cout<<"CIRCLE:\n";
        cout<<"Enter the Radius: ";
        cin>>l;
        a=area(l);
        disp(a);
        cout<<"\n\nTRIANGLE\n:";
        cout<<"Enter the Base and Height:";
        cin>>l>>m;
        a=area(l,m);
        disp(a);
        cout<<"\n\nRECTANGLE:\n";
        cout<<"Enter the Length and Breadth:";
        cin>>m>>n;
        a=area(m,n);
        disp(a);
        getch();
        return 0;
}
```

Operator Overloading:

In **Operator Overloading,** an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. Although semantics of an operator can be extended, but cannot change its syntax, the grammatical rules that govern its use such as the number of operands, precedence and associativity.

For example '+' operator can be overloaded to perform addition on various data types, like for integer, float etc.

Operator overloading is achieved using the **operator function**. The operator function is created using **operator** keyword.

Syntax of operator function:

```
        returntype classname :: operator Op (argument List)
        {
                //Function Body
        }
```

where, **returntype** is the type of value returned by the specified operation.
op is the operator being overloaded. op is preceded by the keyword **operator**. 'operator op' is the function name.
The **argument list** will depend on whether the operator is unary or binary and whether the function is a member function or friend function.

The operator function can be either member function or friend function.

- A friend function will have one argument for unary operators and two for binary operators.
- A member function has no arguments for unary operators and only one for binary operators because the object used to invoke the member function is passed implicitly and therefore is available for member function.

Restrictions on Operator overloading while implementing operator overloading:

1. Precedence and Associativity of an operator cannot be changed.
2. Arity (numbers of Operands) cannot be changed. Unary operator remains unary, binary remains binary etc.
3. No new operators can be created, only existing operators can be overloaded.
4. Cannot redefine the meaning of a procedure. You cannot change how integers are added.
5. There are few operators in C++ that cannot be overloaded such as
    - ternary operator ?:,
    - sizeof,
    - scope resolution operator ::
    - membership operators . and .* .

Overloading Unary operators:
Unary operators are Increment, Decrement and unary minus which can be overloaded.

Example: To overload the operator unary minus- to negate the numbers.

```
#include<iostream.h>
#include<conio.h>
class Numbers
{
        int x;
        int y;
   public:
        void accept(int a,int b)
        {
                x=a;
                y=b;
        }
        void display()
        {
                cout<<"x="<<x<<"\n";
                cout<<"y="<<y<<"\n";
        }
        void operator -()
        {
                x=-x;
                y=-y;
        }
};
int main()
{
        Numbers N;
        clrscr();
        N.accept(10,20);
        N.display();
        -N;
        cout<<"\nAfter unary minus handled variable are:"<<"\n";
        N.display();
        return(0);
}
```

Overloading Unary operators using friend function:

Example: To overload operator unary minus- to negate the numbers using friend function.

```
#include<iostream.h>
#include<conio.h>
class Numbers
{
        int x;
        int y;
   public:
        void accept(int a,int b)
        {
                x=a;
                y=b;
        }
        void display()
        {
                cout<<"x="<<x<<"\n";
                cout<<"y="<<y<<"\n";
        }
        friend void operator -(Numbers &Obj)
        {
                Obj.x=-Obj.x;
                Obj.y=-Obj.y;
        }
};
```

```
int main()
{
        Numbers N;
        clrscr();
        N.accept(10,20);
        N.display();
        operator –(N);
        cout<<"\nAfter unary minus handled variable are:"<<"\n";
        N.display();
        return(0);
}
```

Overloading increment operator:
The increment operator ++ is used in two ways: pre-increment (++d) and post-increment(d++).
To distinguish between pre and post increment operator overloading, dummy parameter of type int in the function heading of the post-increment operator function is used. Decrement operator can be overloaded similarly.

```
void operator++()
{
        ++x;
        ++y;
}
void operator++(int)
{
        x++;
        y++;
}
```

Overloading Binary operators:
Arithmetic operators are most commonly used operator in C++. Almost all arithmetic(+, - , * ,/) operators are overloaded to perform arithmetic operation on user-defined data type.
Example: To Overload Binary '+' operator using member function

```
#include<iostream.h>
#include<conio.h>
class Numbers
{
        int x;
        public:
                void accept(int a)
                {
                        x=a;
                }
                void display()
                {
                        cout<<"x="<<x<<"\n";
                }
```

```
		Numbers operator +(Numbers Obj)
		{
			Numbers temp;
			temp.x=x+Obj.x;
			return temp;
		}
};
int main()
{
	clrscr();
	Numbers N1,N2,N3;
	N1.accept(100);
	N2.accept(200);
	cout<<"\nFirst number:";
	N1.display();
	cout<<"\nSecond number:";
	N2.display();
	cout<<"\nOperations:\n\n";
	cout<<"\nAddition:";
	N3=N1+N2;
	N3.display();
	getch();
	return(0);
}
```

Example: To Overload Binary operator '+' using friend function.

```
#include<iostream.h>
#include<conio.h>
class Numbers
{
	int x;
	public:
		void accept(int a)
		{
			x=a;
		}
		void display()
		{
			cout<<"x="<<x<<"\n";
		}
		friend Numbers operator +(Numbers Obj1,Numbers Obj2)
		{
			Numbers temp;
			temp.x=Obj1.x+Obj2.x;
			return temp;
```

```
                }
};
int main()
{
        clrscr();
        Numbers N1,N2,N3;
        N1.accept(100);
        N2.accept(200);
        cout<<"\nFirst number:";
        N1.display();
        cout<<"\nSecond number:";
        N2.display();
        cout<<"\nOperations:\n\n";
        cout<<"\nAddition using friend function:";
        N3=operator+(N1,N2);
        N3.display();
        getch();
        return(0);
}
```

Overloading insertion and extraction operators:

Overloading insertion(<<) operator and extraction (>>) operator is used to input and output objects using stream class library in the similar way as built in data types.

<< operator is overloaded with **ostream** class object **cout** to print primitive type value to the screen. Similarly **<<** operator is overloaded in class to print user-defined type to screen.

>> operator is overloaded with **istream** class object **cin** to read primitive type values from the user. Similarly **>>** operator is overloaded in class to read user-defined type value.

Example: To Overload << & >> operator.

```
        #include<iostream.h>
        #include<conio.h>
        #include<fstream.h>
        class Numbers
        {
                int x;
          public:
                friend ostream& operator <<(ostream &,Numbers &);
                friend istream& operator >>(istream &,Numbers &);
        };
        ostream & operator <<(ostream &out,Numbers &d)
        {
                out<<"\nValue of x:"<<d.x;
                return out;
        }
        istream & operator >>(istream &in,Numbers &d)
        {
                in>>d.x;
                return in;
        }
        int main()
        {
                Numbers N;
                cout<< "Input";
                cin>>N;                 //invokes operator >>( ) function
                cout<<"Output";
                cout<<N;                //invokes operator <<( ) function

                getch();
                return 0;
        }
}
```

String manipulation using Operator Overloading:

Relational operators like ==, > ,>=, <, <=, !, != are used to compare two user-defined objects.

Example: To compare two strings are equal or not

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class mystring
{
        char str[30];
        int len;
        public:
                mystring(char *s)
                {
                        strcpy(str,s);
                }
                int operator ==(mystring ms)
                {
                        if(strcmp(str,ms.str)==0)
                                return 0;
                        else
                                return 1;
                }
};
int main()
{
        char s1[10],s2[10];
        clrscr();
        cout<<"Enter first string"<<"\n";
        cin>>s1;
        cout<<"Enter second string"<<"\n";
        cin>>s2;
        mystring obs1(s1),obs2(s2);
        if(obs1==obs2)
        cout<<"\nGiven strings are not same"<<"\n";
        else
        cout<<"\nGiven strings are same"<<"\n";
        getch();
        return(0);
}
```

Run time Polymorphism:

An appropriate member function is selected for a particular call while the program is running (at run time) is called as **Run time Polymorphism.** It is also called as late binding or dynamic binding or dynamic linking. **Run time Polymorphism is achieved by using Virtual Function**.

this pointer:

Keyword this is used to represent an object that invokes a member function. this is a pointer that points to the object for which this function was called. This unique pointer is automatically passed to a member function when it is called. The pointer 'this' acts as an implicit argument to all the member functions.

Example1: To illustrate the use of this pointer.

```
#include<iostream.h>
class Test
{
        int x;
public:
        void setX (int x)
        {
                this->x = x;
          }
        void print() { cout << "x = " << x << endl; }
};
int main()
{
        Test obj;
         int x = 20;
         obj.setX(x);
         obj.print();
        return 0;
}
```

Example2: To illustrate the use of this pointer.

```
#include<iostream.h>
#include<conio.h>
class Test
{
             int x;
  public:
            Test(int x)
            {
                        this->x = x;
            }
            Test& maximum(Test& T)
            {
                        if(T.x >= x)
                                    return T;
                        else
                                    return * this;
             }
            void print() { cout << "x = " << x << endl; }
};

int main()
{
   Test obj1(50),obj2(30);
   obj1.print();
   obj2.print();
   Test obj3=obj1.maximum(obj2);
   obj3.print();
   getch();
   return 0;
}
```

Note: return * this will return the object that invoked the function.

Virtual Function:

When same function name is used in both the base and derived classes, the function in base class declared as virtual using the keyword virtual preceding its normal declarations. When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of the pointer. By making the base pointer to point to different objects, different versions of virtual functions can be executed.

Run time Polymorphism is achieved only when a Virtual Function is accessed through a pointer to the base class.

Example: To illustrate the use of virtual function.

```
class Base
{
    public:
        void Display()
        {
                cout<<"\n Display Base";
        }
        virtual void show()
        {
                cout<<"\n Show Base";
        }

};
class Derived: public Base
{
   public:
        void Display()
        {
                cout<<"\n Display Derived";
        }
        void show()
        {
                cout<<"\n Show Derived";
        }
};

int main()
{
        Base B;
        Derived D;
        Base *Bptr;

        Bptr = &B;
        Bptr->Display();                //Calls Base version
        Bptr->Show();                   //Calls Base version

        Bptr = &D;
        Bptr->Display();                //Calls Base version
        Bptr->Show();                   //Calls Derived version

        return(0);
}
```

Note: When Bptr is pointing to derived class object D, the statement

Bptr->Display();

calls only the function associated with Base; whereas the statement

Bptr->Show();

calls the Derived version of Show(). Because Show() function from the base class is declared as virtual.

Pure virtual function:

A virtual function equaled to zero is called as pure virtual function It is also called as **"do-nothing"** function. It is a function declared in a base class that has no definition relative to the base class.

Syntax:

virtual void display()=0;

A class containing such pure function is called as an **abstract class**.

Practice Programs:

1. Write a C++ program to sort integer and float array elements in ascending order by using function overloading.

2. Create a class College containing data members as College_Id, College_Name, Establishment_year, University_Name. Write a C++ program with following member functions:
   i. To accept 'n' College details
   ii. To display College details of a specified University
   iii. To display College details according to a specified establishment year
   (Use Array of Object and Function overloading)

3. Create a class Fraction containing data members as Numerator and Denominator. Write a C++ program to overload operators ++, -- and * to increment, decrement a Fraction and multiply two Fraction respectively. (Use constructor to initialize values of an object).

4. Create a base class Conversion. Derive three different classes Weight (Gram, Kilogram), Volume(Milliliter, Liter), Currency(Rupees, Paise) from Conversion class. Write a C++ program to perform read, convert and display operations. (Use Pure virtual function)

Set A:

1. Write a C++ program to calculate area of cone, sphere and circle by using function overloading.

2. Create a C++ class Employee with data members E_no, E_Name, Designation and Salary. Accept two employees information and display information of employee having maximum salary. (Use this pointer)

3. Write a C++ program to create a class Integer. Write a C++ program to implement necessary member functions to overload the operator unary pre and post decrement '--' for an integer number.

4. Create a C++ class Integer that contains one integer data member. Overload following binary operators (+,-,*, /).

5. Consider a class Point containing x and y coordinates. Write a C++ program to implement necessary functions to accept a point, to display a point and to find distance between two points using operator overloading (-). (Use friend function)

Set B:

1. Create class Person which contains data member as Passport_Id, Person_name, Nationality, Gender, Date_of_Birth, Date_of_Issue, Date_of_expiry. Write a c++ program to perform following member functions:
   i. Enter details of all persons
   ii. Display passport details of one person
   iii. Display passport details of all persons
   (Use Function overloading and Array of object).

2. Create a class Date with members as dd, mm, yyyy. Write a C++ program for overloading operators >> and << to accept and display a Date.

3. Create a class MyString which contains a character pointer (using new operator). Write a C++ program to overload following operators:

i.	<	To compare length of two strings
ii.	!=	To check equality of two strings
iii.	+	To concatenate two strings

4. Create a base class Shape. Derive three different classes Circle, Rectangle and Triangle from Shape class. Write a C++ program to calculate area of Circle, Rectangle and Triangle. (Use pure virtual function).

Set C:

1. Create a class MyString which contains a character pointer (Use new and delete operator).Write a C++ program to overload following operators:

i.	!	To change the case of each alphabet from given string
ii.	[ ]	To print a character present at specified index

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 7: Managing Console I/O Operations

Stream is a sequence of bytes. It represents a device on which input and output operations are performed. C++ provides standard iostream library to operate with streams. The iostream is an object-oriented library which provides Input/Output functionality using streams. C++ stream classes are as follows:

I/O Stream	Meaning	Description
ios	General Input/Output Stream Class	It contains basic facilities that are used by all other input & output classes. Declares constants & functions for handling formatted input & output operations
istream	Input Stream	Inherits properties of ios. It reads and interprets input. Declares input functions get(), getline() and read().Contains overloaded extraction operator >>.
ostream	Output Stream	Inherits properties of ios. It can write sequences of characters and represents other kinds of data. Declares output functions put()and write(). Contains overloaded insertion operator <<.
iostream	Input / Output Stream	Inherits properties of ios istream and ostream & contains all input & output functions.
streambuf	File Stream Base	Provides an interface to physical devices through buffers & acts as a base for filebuf class used ios files

Unformatted I/O Operations:
We have used objects cin & cout which are predefined in iostream file for input & output of various types. cin is an object of type istream & cout is an object of type ostream.
We read data from keyboard using following format:
cin>>variable1>> variable2>>………>> variableN
We write data or display it on screen using following format:
cout<< variable1<< variable2<<………………<< variable

Example: Illustrate use of cin & cout statements.
```
int rno;
cin>>rno;
cout<<”Roll No:”<<rno<<endl;
```
In this we have to study different functions of istream class and of ostream class.

istream class functions:
int get()-Accepts a character from input screen & returns it.
istream &get(char &ch)- Accepts a character from input screen & assigns it to the character ’ch’.
istream &getline(char *buffer,int size,char del=’\n’)-It accepts a string from input stream until it enters a newline character.

ostream class functions:
ostream &put(char ch)- It inserts a character ch in an output screen.
ostream &write(const char *s, streamsize n)- It inserts first n characters of the character array pointed to by ‘s’ into the output screen.

Examples: Program to illustrate use of get() & put() functions.
```
#include<iostream.h>
#include<conio.h>
int main()
{
        char c;
        clrscr();
        cin.get(c); //get a character from keyboard & assigns it to
        cout<<"Entered Character is:"<<c; //display an entered character on output screen.
        return 0;
}
```
In above program we can also display same character entered by user using
cout.put(c) method instead of using cout statement.

Examples: Program to illustrate use of getline() & write() functions.
```
#include<iostream.h>
#include<conio.h>
int main()
{
        int size;
        char name[20];
        char *city="Pune";
        clrscr();
        cout<<"Enter name of student:";//Accept name of student from user
        cin>>name;
        cout<<"Entered name is:"<<name;//Display entered name
        cout<<"City of student is:"<<endl;
        cout.write(city,10);//Display city using write function
        cout<<"Enter another name of student:"<<endl;
        cin.getline(name,size); //Use geline function to accept name
        cout<<"Another name of student is:"<<name; //Display another name of student accepted
        from user
        return 0;
}
```

Formatted Console I/O Operations:
C++ supports a number of features that could be used for formatting output.These features includes:
- ios class functions & flags
- Manipulators
- User-Defined Output functions(Manipulators)

The ios class contains a large number of member functions that help us to format the output in a number of ways.

Manipulators are helping functions that can modify the input or output stream. These format manipulators are available by including the file “<iomanip.h>”.

ios format functions & manipulators:

ios Functions	**Task**	**Equivalent Manipulators**
width()	Specify required field size for displaying an output value	setw()
precision()	Specify number of digits to be displayed after the decimal point of float value	setprecision()
fill()	Specify a character that is used to fill the unused portion of a field	setfill()
setf()	Specify format flags that can control the form of output display i. e left justification & right justification	setiosflags()
unsetf()	To clear the flags specified	resetiosflags()

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
int main()
{
        clrscr();
        cout.width(5);              //set width to 5
        cout<<123<<12<<endl;        //dispay output in width of box 5
        cout.width(5);
        cout<<543;
        cout.width(5);
        cout<<19<<endl;
        cout.precision(3);          //display 3 digits after decimal point
        cout<<sqrt(2)<<"\n";
        cout.precision(4);          //display 4 digits after decimal point
        cout<<sqrt(3)<<"\n";
        cout.fill('*');             //Padding fill with '*'
        cout.width(10);
        cout<<"SYBBA"<<"\n";
        cout.fill('#');             //Padding fill with'#'
        cout.setf(ios::right,ios::adjustfield);    //it display output to right side
        cout.width(12);
        cout<<"CA"<<"\n";
        return 0;
}
```

Output:

```
  12312
```

```
543   19
1.414
1.7321
*****SYBBA
#########CA
```

User-Defined Manipulators:
In addition to predefined functions C++ allows us to create our own manipulator functions to provide any special output formats.

```
ostream & manipulator(ostream & output)
{
        ----------
        // code
        ----------
        return output;
}
```
manipulator is the name of manipulator under creation.

Example: Program to illustrate how to create user defined manipulator.
```
#include<iostream.h>
ostream & unit(ostream & output)
{
        output<<"Kilograms";
        return output;
}
int main()
{
        cout<<"Weight:"<<40<<unit;
        return 0;
}
```
Output:

```
C:\TURBOC3\BIN>TC
Weight:40Kilograms
```

Practice programs:
1. Define a class Item that contains data member as Item_no, Item _Name, Item _Price. Derive a class Discount(discount_in_percentage) from class Item. A Customer buys 'n' items. Accept quantity for each item, calculate total discount and accordingly generate and display the bill using appropriate Manipulators.

Set A:
1. Write a C++ program to create a class Employee which contains data members as Emp_Id, Emp_Name, Basic_Salary, HRA, DA, Gross_Salary. Write member functions to accept Employee information. Calculate and display Gross salary of an employee.

(DA=25% of Basic salary and HRA = 40% of Basic salary) (Use appropriate manipulators to display employee information in given format :- Emp_Id and Emp_Name should be left justified and Basic_Salary, HRA, DA, Gross salary Right justified with a precision of three digits)

2. Write a C++ program to create a class Teacher which contains data members as Teacher_Name, Teacher_City, Teacher_Contact_Number. Write member functions to accept and display five teachers information. Design User defined Manipulator to print Teacher_Contact_Number. (For Contact Number set right justification, maximum width to 10 and fill remaining spaces with '*')

Set B:

1. Create a C++ class Train with data members as Train_No,Train_Name,No_of Seats,Source_Station,Destination_Station. Write necessary member functions for the following:
   i. Accept details of n trains.
   ii. Display all train details.
   iii. Display details of train from specified starting station and ending station by user.
2. Create a C++ class Manager with data members Manager_Id, Manager_Name, Mobile_No., Salary. Write necessary member functions for the following:
   i. Accept details of n managers
   ii. Display manager details in ascending order of their salary.
   iii. Display details of a particular manager. (Use Array of object and Use appropriate manipulators.)

Set C:

1. Create a C++ class Marksheet with data members Seat_No., Student_Name, Class, Subject_Name, Int_Marks, Ext_Marks, Total, Grand_Total, Percentage, Grade. Write member function to accept Student information for 4 subjects. Calculate Total, Grand_Total, Percentage, Grade and display Marksheet. (Use user defined manipulator)

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 8: Working with Files

File: It is collection of data or information. **Stream**: It is sequence of bytes.

To perform input and output operations on files, three classes included in the <fstream.h> library. It defines several classes including ifstream , ofstream and fstream.

Stream	**Description**
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.

Opening file:
File can be opened by using member function open() or by using constructor.

Stream	**Description**	**Examples By Using Constructor**	**Examples By Using Member Function**
ofstream	Stream class to write on files	ofstream outfile1("first.txt");	ofstream outfile2; outfile2.open("second.txt");
ifstream	Stream class to read from files	ifstream infile1("first.txt");	ifstream infile2; infile2.open("second.txt");
fstream	Stream class to both read and write to / from files.	fstream file1("first.txt",ios::out);	fstream file2; file2.open("second.txt",ios::out);

Detecting End-Of-File:
It checks whether end of file occurs or not. eof() is member function of ios class. It returns nonzero value if end of file condition is encountered and zero otherwise.

Syntax:
```
ifstream fin;
if(fin.eof()!=0)
{
        exit(1);
}
```
This statement terminates the program on reaching end of file.

File Opening Modes:
There are different modes (flags) of a file which are listed below:

Parameter	**Meaning**
ios::in	Open for input operations.
ios::out	Open for output operations.

ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunk	If the file is opened for output operations and it already exists, its previous content is deleted and replaced by the new one.

Closing File:
A file which is opened while reading or writing in **file handling** must be closed after performing an action on it.

Syntax:
filename.close();

File Pointer and Their Manipulations:
Each file has two pointers associated with it known as file pointers.
Input pointer (get pointer)
Output pointer (put pointer)

Following member functions are used to move the file pointer at the desired position while reading or writing from the file.

Function	**Description**
seekg()	Moves get pointer(input) to specified location.
seekp()	Moves put pointer(output) to a specified location.
tellg()	Gives the current position of the get pointer.
tellp()	Gives the current position of the put pointer.

File Handling Functions:
C++ provides us with the following operations in File Handling:

open() Function-To create a file by using open function.
file.open("sample.txt",ios::in |ios::out);

get() & put() Functions- put() writes a single character in file and get() reads a single character from a file.
fstream file; //Input & output stream
file.put('h'); //put char to file
file.get(ch); //get character from file

read() & write() Functions- These functions are used to perform read & write operations on binary file.
infile.read((char *) & v, sizeof(v));
infile.write((char *) & v, sizeof(v));

These functions take two arguments. The first is address of the variable V and second is the length of that variable in bytes.

Text and Binary files:
The C++ language supports two types of files:

- Text files
- Binary files

Text Files:
These files are designed to store text. In such files various character translations are performed such as “\r+\f” is converted into “\n”, whereas in binary files no such translations are performed. By default, C++ opens the files in text mode.
Example: Program to illustrate reading & writing to text file .
```
#include<fstream.h>
int main ()
{
  char sname[20]="SYBBA",line[20];
  ofstream outfile;
  outfile.open("example.txt");
  outfile<<sname;
  outfile.close();
  ifstream infile;
  infile.open("example.txt");
  infile.getline(line,20);
  cout<<line;
  infile.close();
  return 0;
}
```
OUTPUT:
```
SYBBA
```
By using above program we can write ‘SYBBA’ to text file ’example.txt’. Then we perform read operation on same file & display ‘SYBBA’ to output screen

Binary Files:
It is used to read & write a given number of bytes on the given stream. write() is a member function of ostream inherited by ofstream and read is a member function of istream inherited by ifstream. Objects of class fstream have both.

Example: Program to illustrate reading & writing to Binary File.
```
// writing on a text file
#include<fstream.h>
int main ()
{
  char sname[20]="SYBBA";
  ofstream outfile;
  outfile.open("example.bin");
  outfile.write((char *) & sname,sizeof(sname));
  outfile.close();
  ifstream infile;
  infile.open("example.txt");
  infile.read((char *)& sname,sizeof(sname));
  cout<<sname;
  infile.close();
  return 0;
}
```

OUTPUT

```
SYBBA
```

By using above program we can write ‘SYBBA’ to text file ’example.bin. Then we perform read operation on same file & display ‘SYBBA’ to output screen

Reading & Writing Class Objects:
How class objects can be written to & read from disk files.

Example: Program to illustrate reading & writing class objects.

```
#include <iostream.h>
#include <fstream.h>
class student
{
        int rno;
        char name[20];
        public:
        void getdata();
        void putdata();
};
void student:: getdata()
{
        cout<<"Enter rno:\n";
        cin>>rno;
        cout<<"Enter name:\n";
        cin>>name;
}
void student:: putdata()
{
         cout<<"Roll No:"<<rno<<endl;
         cout<<"Name:"<<name<<endl;
}
int main ()
 {
   student s[3];
   fstream file;
   file.open("student.txt",ios::in | ios::out);
```

```
    cout<< "Enter details of 3 students:\n";
    for(int i=0;i<3;i++)
    {
            s[i].getdata();
            file.write((char *) & s[i],sizeof(s[i]));
    }
    file.seekg(0); //reset to start
    cout<<"\nOUTPUT\n\n";
    for(i=0;i<3;i++)
    {
            file.read((char *) & s[i],sizeof(s[i]));
            s[i].putdata();
    }
    file.close();
    return 0;
}
```

OUTPUT:

```
Enter details of 3 students:
Enter rno:1
Enter name:Avani
Enter rno:2
Enter name:Ananya
Enter rno:3
Enter name:Kavya

OUTPUT
Roll No:1
Name:Avani
Roll No:2
Name:Ananya
Roll No:3
Name:Kavya
```

Updating A File:Random Access

Updating is the maintenance of any data file. The updating includes one or more of following tasks:

- Displaying contents of a file
- Modifying an existing item
- Adding a new item
- Deleting an existing item

These action requires the file pointers to move to a particular location. File contains collection of items of equal lengths. Size of each item/object can be obtained using

```
int object_length=sizeof(object);
```

Location of object can be obtained using int location=m* object_length;
This location gives us byte number of the first byte of mth object.We can set file pointer to reach this byte with the help of seekg() & seekp() .

We also find total number of objects ina file using object_length as follows:
int n=file_size/ object_length;
The file_size can be obtained using function tellg() & tellp() when file pointer is located at the end of file.

Error Handling During File Operations:
There are several error handling functions supported by class ios that help you read and process the status recorded in a file stream. Following table lists these error handling functions and their meaning :

Function	**Meaning**
int eof()	Returns non-zero (true value) if end-of-file is encountered while reading; otherwise returns zero (false value).
int fail()	Returns non-zero (true) when an input or output operation has failed.
int bad()	Returns a non-zero value if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is zero (false value), it may be possible to recover from any other error reported and continue operations.
int good()	Returns non-zero (true) if no error has occurred. This means, all the above functions are false. For example, if fin.good() is true, everything is okay with the stream named as fin and we can proceed to perform I/O operations. When it returns zero, no further operations can be carried out.

Command Line Arguments:
We supply arguments to main function at the time of invoking program by command line argument. They may be used to pass the names of data files.
Example: Program to illustrate use of command line arguments.

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
int number[9]={1,2,3,4,5,6,7,8,9};
if(argc!=2)
{
        cout<<"argc="<<argc<<"\n";
        cout<<"Error in arguments\n";
        exit(1);
}
ofstream fout1;
fout1.open(argv[1]);
if(fout1.fail())
{
        cout<<"Unable to open a file"<<argv[1]<<"\n";
        exit(1);
}
else
{
        for(int i=0;i<9;i++)
        {
        if(number[i]%2==0)
        fout1<<number[i]<<""; //write all even numbers from number array to file
        }
}
fout1.close();
ifstream fin;
int i;
char ch;
for(i=1;i<argc;i++)
{
        fin.open(argv[i]);
        cout<<"Contents of"<<argv[i]<<"\n";
        do
        {
        fin.get(ch);//reads an even numbers from file
        cout<<ch; //display it
}while(fin);
cout<<"\n\n";
fin.close();
}
return 0;
}
```

Output:

```
C:\TC\SOURCE>temp a.txt
Contents of a.txt
2468
C:\TC\SOURCE>exit
```

To run this program we first compile it. Then instead of using Ctrl+F9 we have to run it by dos shell. Click on File & then select DOS Shell.DOS Shell gets opened. Now give program name space a.txt. All even numbers will get added to a.txt file afterwards we print even numbers from 'a.txt' file to the output screen. Sometime program name is not found in BIN directory so change directory from BIN to SOURCE & then run your program.

Practice programs:

1. Write a C++ program to copy even numbers from the file “Numbers.txt” into the file “even.txt” and odd numbers into the file “odd.txt”. Display the count of numbers in each file. Compute the median and average of numbers in both files.
2. Write a C++ program that reads a “source.txt” file and creates another file named as “destination.txt” which is identical to source except that every sequence of consecutive blank spaces is replaced by a single space.
3. Write a C++ program to read the contents from the file “sample.txt”. Store all the characters from “sample.txt” into the file “character.txt” & store all digits into the file “digit.txt “.
4. Write a C++ program which will accept ‘n’ integers from user through command line argument. Store Prime numbers in file “Prime.txt” and remaining numbers in “Others.txt”.

Set A:

1. Write a C++ program to accept ‘n’ numbers from user through Command Line Argument. Store all positive and negative numbers in file “Positive.txt” and “Negative.txt” respectively.
2. Write a C++ program to read the contents of a text file. Count and display number of characters, words, lines and blank spaces from a file. Find the number of occurrences of a given word present in a file.
3. Create a C++ class Employee with data members Emp_No, Emp_Name, Emp_Marks. Write necessary member functions for the following:
   i. Accept the details and store it into the file “Emp.dat”
   ii. Read the details from file and display it.
   iii. Update a given record into the file.

Set B:

1. Write a C++ program to create a class Newspaper with data members Name, publisher, cost. Write necessary member functions for the following:
   i. Accept details for ‘n’ Newspapers from user and store it in a file “Newspaper.txt”.
   ii. Display details of Newspapers from a file.
   iii. Count the number of objects stored in a file.
2. Write a C++ program that reads from a formatted file a list of 4 students and their marks for 3 tests, computes the average test score for each student and the grade and outputs them in another file.
3. Create a C++ class ‘city’ with data members name and STD code. Accept ‘n’ cities with STD codes from user. Store this data in the file ‘cities.txt’. Write a program that reads the data from file cities.txt display the list of city with STD codes from a file

Set C:

1. Create a C++ class MyFile containing:

   fstream fp;
   Char *fn;
   Write necessary member Functions using operator overloading:

+	F3=F1+F2	Put contents of F1 and F2 in F3.
-	-F3	Changes the case of all upper and lower case characters in F3.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 9: Templates

Templates are powerful features of C++ which allows you to write generic programs. 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. In simple terms, we can create a single function or a class to work with different data types using templates. Templates are often used in larger code base for the purpose of code reusability and flexibility of the programs. The concept of templates can be used in two different ways:

- Function Templates
- Class Templates

Function Template:
It is used to define generic functions. A single function template can work with different data types at once. It works on different types of data.

Syntax of function template with single parameter:
A function template starts with the keyword template followed by template parameter(s) inside <> which is followed by function declaration.

```
template <class T>
returntype functionName(arguments of type T)
{
   // Body of function with type T
}
```

T is a generic name for a data type used by the function. This name can be used within the function definition.

Syntax of function template with Multiple Parameters:

```
template <class T1,class T2>
returntype functionName(arguments of types T1,T2,………….)
{
   // Body of function
}
```

Example: Program to illustrate use of Function Template with multiple parameters.

```
#include<iostream.h>
template<class T>
T add(T num1, T num2)
{
   return (num1 + num2);
}

int main()
{
   int result1;
```

```
    double result2;
    // calling with int parameters
    result1 = add(2, 3);
    cout << "2 + 3 = " << result1 << endl;

    // calling with double parameters
    result2 = add(2.2, 3.3);
    cout << "2.2 + 3.3 = " << result2 << endl;

    return 0;
}
```

Output:

```
2 + 3 = 5
2.2 + 3.3 = 5.5
```

Class Template:

Class templates are used for writing generic class operations. We would need to create a different class for each data type or create different member variables and functions within a single class using a class template.

Syntax of class template with single parameter:

```
template <class T>
class className
{
          public:
            ... .. ...
          //class member specification with anonymous type T
            ... .. ...

            ... .. ...

};
```

T is a generic name for a data type which will be specified when a class is instantiated. we can define more than one generic data type by using a comma-separated list.

Syntax of class template with multiple parameters:

```
template <class T1, class T2,…..>
class className
{
          public:
            ... .. ...
            ... .. ...
            ... .. ...

};
```

Example: Program to illustrate use of Class Template with multiple parameters.

```
#include<iostream.h>
#include<conio.h>
template<class T1, class T2>
class A
{
  T1 a;
  T2 b;
   public:
  A(T1 x,T2 y)
{
   a = x;
   b = y;
}
void display()
{
   cout<<"Values of a and b are :"<< a<<","<<b<<endl;
}
};
int main()
{
        clrscr();
        A<int,float> d(5,6.5);
        d.display();
        return 0;
}
```

Output:

```
Values of a and b are: 5, 6.5
```

Practice programs:

1. Write a C++ program to swap two integer values and two float values by using function template.

Set A:

1. Write a C++ template program to accept array elements of type integers & characters. Reverse an array of both types.
2. Write a C++ program to find maximum & minimum of two integer numbers and two float numbers by using function template.
3. Write a C++ template program to sort the array elements of type integer, float and character.

Set B:

1. Write a C++ program to define class template for calculating the square of given numbers with different data types.

2. Write C++ template program to find the area of circle & rectangle with different data types.
3. Write a template to represent a generic vector. Include member functions to perform the following tasks:
    i. To create the vector.
    ii. To modify the value of a given element.
    iii. To multiply the vector by a scalar value.
    iv. To display the vector in the form (10, 20, 30,…..)

Set C:

1. Write C++ template program to implement stack & its operations like push & pop.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Section-II

NODE JS

ASSIGNMENT NO. 1: NODE.JS WEB SERVER, MODULES & NPM

Introduction:

Node.js is an open-source server-side runtime environment that provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.

Node.js can be used to build different types of applications such as command line application, web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET.

Downloads, Installation and setting up Environment for node.js

- The official Node.js website has installation instructions for Node.js: https://nodejs.org
- Download Editor visual studio code for node js from: https://code.visualstudio.com/download
- Once you have downloaded and installed Node.js & VS code editor on your computer, you can run "Hello World" node.js app and display Hello World! Message on a web browser.
- Create a Node.js file named "myfirst.js", and add the following code:

  myfirst.js

```
var http = require('http');
http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/html'});
res.end('Hello World!');
}).listen(8081);
console.log('Server running at http://127.0.0.1:8081/');
```

Now execute the myfirst.js to start the server as follows −

- **$ node myfirst.js**
- Verify the Output on browser. Server has started.
- Server running at http://127.0.0.1:8081/

  A Node.js application consists of the following three important components −

  Import required modules − We use the require directive to load Node.js modules.

  Create server − A server which will listen to client's requests similar to Apache HTTP Server.

  Read request and return response − The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

  Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

  To include the HTTP module, use the require() method:

  var http = require('http');

Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.Use the createServer() method to create an HTTP server:

REPL Terminal:

- REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode.
- Node.js or Node comes bundled with a REPL environment. It performs the following tasks -
- Read − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- Eval − Takes and evaluates the data structure.
- Print − Prints the result.
- Loop − Loops the above command until the user presses ctrl-c twice.
- The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

  REPL can be started by simply running node on shell/console without any arguments as follows.

```
$ node
```

You will see the REPL Command prompt > where you can type any Node.js command −

```
$ node > Simple Expression
```

Let's try a simple mathematics at the Node.js REPL command prompt −

```
$ node > 1 + 3
  4
> 1 + ( 2 * 3 ) - 4
3
```

You can make use variables to store values and print later like any conventional script. If var keyword is not used, then the value is stored in the variable and printed. Whereas if var keyword is used, then the value is stored but not printed. You can print variables using console.log().

```
$ node > x = 10 10
> var y = 10 undefined
> x + y
  20
> console.log("Hello World")
Hello World undefined
```

Modules in Node.js

Node.js has a simple module loading system. A module in node.js is a simple or complex functionality organized in single or multiple javascript files which can be reusable again.

Node.js Module Types
In Node.js modules can be categorized in 3 types

1. Core Modules
2. Local Modules
3. Third Party Modules

Node.js Core Modules

Node.js has several modules compiled into its binary distribution, and load automatically when the Node.js process starts, these are called the core modules. These core modules of node.js are located within Node.js's source and are located inside **"lib"** folder.

Some of the core modules are listed below.

- ***http*** - This module is used to create http server.
- ***fs*** - This module is used to perform file operations like reading, writing, appending and deleting files etc.
- ***Crypto*** - This module provides cryptographic functionalities like encryption, decryption, sign, verification, digesting etc.
- ***Querystring*** - This method includes methods to deal with querystring like unescapeBuffer, unescape, escape, encode, stringify, decode and parse.
- ***url*** - This module includes methods for url resolutions,resolve, parsing, format etc.
- *path* - This module is used to deal with file paths when working with file system.

Local Modules
Local modules are user defined modules which are mainly used for specific projects and locally available in separate files or folders within project folders. These types of modules contain application specific functionality.
Note
We can package locally created local modules and distribute them via NPM (Node Package Manager), which can be used by others and the node community.

Third party module
The third-party module can be downloaded by NPM (Node Package Manager). These types of modules are developed by others and we can use that in our project. Some of the best third party module examples are: express, gulp, lodash, async, socket.io, mongoose, underscore, pm2, bower, q, debug, react, mocha etc.

Third party modules can be install inside the project folder or globally.

How to load a module?

To load a module in your node application you can just use "require()" function. whose syntax is given below.

var module=require('module_name');

There are several ways to reference modules, this depends on what type of module we are going to load.

Loading core module

Core modules can be loaded as follows.

```
var http=require('http');
```

As I have already told you that code modules are loaded in "lib" directory, so in the above example http module will be loaded from lib folder.

How to create and load local module?

In Node.js files and modules are in a one-to-one correspondence. The following example will explain to you how to create a local Node.js module.

```
function Circle(radius) {
 return {
  area: function area() {
  return Math.PI * Math.PI * radius;
   }
       };
}
module.exports = Circle;
```

In the above example We have created a function which is used to find an area of the Circle. In this example in the last line I have written "module.exports=Circle" this is a very important line here. Here module is a variable that represents the modules in which we are currently in. We can export any type of object. Save above file using "Circle.js". To use this module inside another file, app.js, the code can be written as follows.

app.js	**Output**

var circle=require('./Circle.js');
var obj=circle(7);
var output=obj.area();
console.log(output);
	C:\Users\Dell-PC\Node-app> node app.js
	69.0872308076255

In the above example We loaded a file whose name is Circle.js using the require function which exports Circle object.

Different ways for loading local node.js module

There are lots of ways to load locally created modules.

- Using absolute path
  var module = require('/<folder_name1>/<folder_name2>/.../module');
- Using relative path
  var module = require('./module');

Here We are not giving .js extension so there is no need to add ".js" Node finds .js files if we do not give .js as an extension, it means the following lines will be the same.

1. **var module = require('./module');**
2. **var module = require('./module.js');**

Using folder path

You can also use folder path to load modules as follows:

```
var module = require('./folder_name');
```

But a folder can contain lots of modules and javascript files so node finds index.js file and loads by default. Otherwise we can create package.json where we can define node module name which we want to load by default. We can write package.json like as follows.

```
{
 "name" : "module_name",
 "main" : "./folder_path/module_name.js"
}
```

Loading third party modules

Third party Node.js module can be downloaded using NPM (node package manager) which you can download locally or globally. To download globally we use the following command.

```
npm install -g <module_name>
```

here we use -g to install package globally. If you want to install locally then use the following command.

```
npm install --save <module_name>
```

Above command will download node package inside node_modules folder and then you can directly use require function to load node module.

```
var module= require('module_name');
```

Caching Modules

In Node.js modules are cached when module is loaded the first time. It means if you load the same node module 2 times then node.js does not load that module again it will copy that module from cache. Example is shown below.

```
console.log("Module Loaded Successfully");
```

We have created a module that is My_Module and written the above code and saved it using My_Modules.js. Now I am creating app.js file where I am writing the following script.

```
var my_module1=require('./My_Module');
```

Above code will print the following output.

```
C:\Users\Dell-PC\Node-app> node app.js
   Module Loaded Successfully
```

Now We are modifying app.js and writing the following code.

```
var my_module1=require('./My_Module');
var my_module2=require('./My_Module');
```

In the above code We have created two objects for My_Module. But when you run it then you will get the following output.

```
C:\Users\Dell-PC\Node-app> node app.js
   Module Loaded Successfully
```

This means that module initializes only once. This is very important to know if you are creating any module.

Practice Programs:

1) Create a Node.js Application that performs following operations on Buffer data.
   1. Concat
   2. Compare
   3. Copy
2) Create node.js application that uses local module to find age of person after accepting date of birth.
3) Create node.js application that create navigation bar on your web page and on selecting option from navbar, goes to respective page such as contact us, about us etc.
4) Create node.js application that create unit conversion module such as mm to cm and using it perform conversion.

SET A

1) Create a Node.js file that will convert the output "Hello World!" into upper-case letters.
2) Create a Node.js Application that uses user defined Module to return the Sum of digits of a of given number.
3) Create a Node.js Application that uses user defined module circle.js which exports the functions area () and circumference () and display the details on console.

SET B

1) Create a Node.js Application that accepts first name, last name of a Person and define a Module that concatenate first name and last name.
2) Create a Node.js Application that uses user defined Module to return the Factorial of a of given number.
3) Create Node js application using user defined Rectangle module to find area of rectangle and display the details on console.

SET C

1) Create Nodejs Module and Publish Over to npm

Signature of the instructor: _ _ _ Date: _

Assignment Evaluation:

0: Not Done	1: Incomplete	2: Late Complete	3: Needs Improvement
4: Complete	5: Well-Done		

ASSIGNMENT NO. 2: FILE SYSTEM

Node.js as a File Server

The Node.js file system module allows you to work with the file system on your computer.
To include the File System module, use the require() method:

```
var fs = require('fs');
```

Common use for the File System module:

- Read files
- Write files
- Create files
- Update files
- Delete files
- Rename files

Major File I/O methods.

Read Files

The fs.readFile() method is used to read files on your computer.
Assume we have the following HTML & demo_readfile.js file (located in the same folder as Node.js):

demofile1.html

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

demo_readfile.js

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8081);
```

Initiate demo_readfile.js:
C:\Users\Your Name>node demo_readfile.js
you will see the result on browser with url: http://localhost:8081

Synchronous vs Asynchronous

Every method in the fs module has synchronous as well as asynchronous forms. Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error. It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, whereas the second one does.

Example

Create a text file named **input1.txt** with the following content −
We are students of SY BBA (CA)
learning node.js in simple and easy way!!!!!

Let us create a js file named **main.js** with the following code & **run the main.js** to see the result:

```
var fs = require("fs");
// Asynchronous read
fs.readFile('input1.txt', function (err, data) {
  if (err) {
    return console.error(err);
  }
  console.log("Asynchronous read: " +
data.toString());
});

// Synchronous read
var data = fs.readFileSync('input1.txt');
console.log("Synchronous read: " +
data.toString());
console.log("Program Ended");
```

$ node main.js

Output:

```
Synchronous read: We are students of SY
BBA (CA)
learning node.js in simple and easy way!!!!!

Program Ended
Asynchronous read: We are students of SY
BBA (CA)
learning node.js in simple and easy way!!!!!
```

Open a File
Syntax
Following is the syntax of the method to open a file in asynchronous mode −
fs.open(path, flags[, mode], callback)
Parameters

Here is the description of the parameters used −

- **path** − This is the string having file name including path.
- **flags** − Flags indicate the behavior of the file to be opened. All possible values have been mentioned below.
- **mode** − It sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.
- **callback** − This is the callback function which gets two arguments (err, fd).

Flags

Flags for read/write operations are −

Sr.No.	**Flag & Description**
1	r: Open file for reading. An exception occurs if the file does not exist.
2	r+: Open file for reading and writing. An exception occurs if the file does not exist.
3	rs: Open file for reading in synchronous mode.
4	Open file for reading and writing, asking the OS to open it synchronously. See notes for 'rs' about using this with caution.
5	Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
6	Wx: Like 'w' but fails if the path exists.
7	Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
8	wx+: Like 'w+' but fails if path exists.
9	a: Open file for appending. The file is created if it does not exist.
10	ax: Like 'a' but fails if the path exists.

11	a+: Open file for reading and appending. The file is created if it does not exist.
12	ax+: Like 'a+' but fails if the the path exists.

Example

Let us create a js file named main.js having the following code to open a file input.txt for reading and writing.

```
var fs = require("fs");
// Asynchronous - Opening File
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
});
```

Now run the main.js to see the result −

$ node main.js

Output:

```
Going to open file!
File opened successfully!
```

Syntax

Following is the syntax of the method to get the information about a file −

fs.stat(path, callback)

Parameters:

Here is the description of the parameters used −

- **path** − This is the string having file name including path.
- **callback** − This is the callback function which gets two arguments (err, stats) where stats is an object of fs.Stats type which is printed below in the example.

Apart from the important attributes which are printed below in the example, there are several useful methods available in fs.Stats class which can be used to check file type. These methods are given in the following table.

Sr.No.	Method	Description
1	stats.isFile():	Returns true if file type of a simple file.
	stats.isDirectory():	Returns true if file type of a directory.
3	stats.isBlockDevice():	Returns true if file type of a block device.
4	stats.isCharacterDevice():	Returns true if file type of a character device.
5	stats.isSymbolicLink():	Returns true if file type of a symbolic link.
6	stats.isFIFO():	Returns true if file type of a FIFO.
7	stats.isSocket():	Returns true if file type of asocket.

Example

Let us create a js file named main.js with the following code –

main.js

```
var fs = require("fs");
console.log("Going to get file info!");
fs.stat('input.txt', function (err, stats) {
  if (err) {
    return console.error(err);
  }
  console.log(stats);
  console.log("Got file info successfully!");
  // Check file type
  console.log("isFile ? " + stats.isFile());
  console.log("isDirectory ? " +
stats.isDirectory());
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to get file info!
Stats {
  dev: 3666283250,
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 281474976790862,
  size: 83,
  blocks: 0,
  atimeMs: 1611035707736.8445,
  mtimeMs: 1611034982031.1924,
  ctimeMs: 1611034982031.1924,
  birthtimeMs: 1609511475673.6436,
  atime: 2021-01-19T05:55:07.737Z,
  mtime: 2021-01-19T05:43:02.031Z,
  ctime: 2021-01-19T05:43:02.031Z,
  birthtime: 2021-01-01T14:31:15.674Z
}
Got file info successfully!
isFile ? true
isDirectory ? false
```

Writing a File

Syntax : Following is the syntax of one of the methods to write into a file −

fs.writeFile(filename, data[, options], callback)

This method will over-write the file if the file already exists. If you want to write into an existing file then you should use another method available.

Parameters : Here is the description of the parameters used −

- **path** − This is the string having the file name including path.
- **data** − This is the String or Buffer to be written into the file.
- **options** − The third parameter is an object which will hold {encoding, mode, flag}. By default. encoding is utf8, mode is octal value 0666. and flag is 'w'
- **callback** − This is the callback function which gets a single parameter err that returns an error in case of any writing error.

Example : Let us create a js file named main.js having the following code –

```
var fs = require("fs");
console.log("Going to write into existing file");
fs.writeFile('input2.txt', 'Simply Easy Learning!',
function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("Data written successfully!");
  console.log("Let's read newly written data");

  fs.readFile('input2.txt', function (err, data) {
    if (err) {
      return console.error(err);
    }
    console.log("Asynchronous read: " + data.toString());
  });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to write into existing file
Data written successfully!
Let's read newly written data
Asynchronous read: Simply
Easy Learning!
```

Syntax

Following is the syntax of one of the methods to read from a file −

fs.read(fd, buffer, offset, length, position, callback)

This method will use file descriptor to read the file. If you want to read the file directly using the file name, then you should use another method available.

Parameters

Here is the description of the parameters used −

- **fd** − This is the file descriptor returned by fs.open().
- **buffer** − This is the buffer that the data will be written to.
- **offset** − This is the offset in the buffer to start writing at.
- **length** − This is an integer specifying the number of bytes to read.
- **position** − This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.
- **callback** − This is the callback function which gets the three arguments, (err, bytesRead, buffer).

Example

Let us create a js file named main.js with the following code −

main.js

```
var fs = require("fs");
var buf = Buffer.alloc(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
  console.log("Going to read the file");

  fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
    if (err){
      console.log(err);
    }
    console.log(bytes + " bytes read");

    // Print only read bytes to avoid junk.
    if(bytes > 0){
      console.log(buf.slice(0, bytes).toString());
    }
  });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to open an existing file
File opened successfully!
Going to read the file
83 bytes read
Going to open an existing file
File opened successfully!
```

Closing a File

Syntax

Following is the syntax to close an opened file –

fs.close(fd, callback)

Parameters

Here is the description of the parameters used −

- **fd** − This is the file descriptor returned by file fs.open() method.
- **callback** − This is the callback function No arguments other than a possible exception are given to the completion callback.

Example

Let us create a js file named main.js having the following code –

main.js

```
var fs = require("fs");
var buf = new Buffer.alloc(1024);

console.log("Going to open an existing file");
fs.open('input2.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
  console.log("Going to read the file");

  fs.read(fd, buf, 0, buf.length, 0, function(err, bytes) {
    if (err) {
      console.log(err);
    }

    // Print only read bytes to avoid junk.
    if(bytes > 0) {
      console.log(buf.slice(0, bytes).toString());
    }

    // Close the opened file.
    fs.close(fd, function(err) {
      if (err) {
        console.log(err);
      }
      console.log("File closed successfully.");
    });
  });
});
```

Now run the main.js to see the result –

```
$ node main.js
```

Output:

```
Going to open an existing file
File opened successfully!
Going to read the file
Simply Easy Learning!
File closed successfully.
```

Update Files

The File System module has methods for updating files:

- fs.appendFile()
- fs.writeFile()

The fs.appendFile() method appends the specified content at the end of the specified file:

Example
Append "This is my text." to the end of the file "input2.txt":

main.js	Now run the main.js to see the result −

```
var fs = require('fs');
fs.appendFile('input2.txt', ' This is my text.',
function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```
	$ node main.js
	
	Output:
	Updated!

Truncate a File
Syntax:
Following is the syntax of the method to truncate an opened file –

fs.ftruncate(fd, len, callback)

Parameters:

Here is the description of the parameters used −

- **fd** − This is the file descriptor returned by fs.open().
- **len** − This is the length of the file after which the file will be truncated.
- **callback** − This is the callback function No arguments other than a possible exception are given to the completion callback.

Example

Let us create a js file named main.js having the following code –

main.js	

```
var fs = require("fs");
var buf = Buffer.alloc(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to truncate the file after 10 bytes");

   // Truncate the opened file.
   fs.ftruncate(fd, 10, function(err) {
      if (err) {
         console.log(err);
      }
      console.log("File truncated successfully.");
```
	Now run the main.js to see the result −
	$ node main.js
	Output:
	Going to open an existing file
	File opened successfully!
	Going to truncate the file after 10 bytes
	File truncated successfully.
	Going to read the same file
	Simply Eas
	File closed successfully.

```
      console.log("Going to read the same file");

      fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
         if (err) {
            console.log(err);
         }

         // Print only read bytes to avoid junk.
         if(bytes > 0) {
            console.log(buf.slice(0, bytes).toString());
         }

         // Close the opened file.
         fs.close(fd, function(err) {
            if (err) {
               console.log(err);
            }
            console.log("File closed successfully.");
         });
      });
   });
});
```

Delete a File

Syntax: Following is the syntax of the method to delete a file –

fs.unlink(path, callback)

Parameters

Here is the description of the parameters used −

- path − This is the file name including path.
- callback − This is the callback function No arguments other than a possible exception are given to the completion callback.

Example : Let us create a js file named main.js having the following code –

Main.js

```
var fs = require("fs");
console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to delete an existing file
File deleted successfully!
```

Create a Directory

Syntax:

Following is the syntax of the method to create a directory –

fs.mkdir(path[, mode], callback)

Parameters:

Here is the description of the parameters used −

- path − This is the directory name including path.
- mode − This is the directory permission to be set. Defaults to 0777.
- callback − This is the callback function No arguments other than a possible exception are given to the completion callback.

Example

Let us create a js file named main.js having the following code –

main.js

```
var fs = require("fs");
console.log("Going to create directory /tmp/test");
fs.mkdir('/tmp/test',function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("Directory created successfully!");
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to create directory /tmp/test
Directory created successfully!
```

Read a Directory

Syntax: Following is the syntax of the method to read a directory –

fs.readdir(path, callback)

Parameters:

Here is the description of the parameters used −

path − This is the directory name including path.

callback − This is the callback function which gets two arguments (err, files) where files is an array of the names of the files in the directory excluding '.' and '..'.

Example

Let us create a js file named main.js having the following code –

main.js

```
var fs = require("fs");
console.log("Going to read directory /tmp");
fs.readdir("/tmp/",function(err, files) {
  if (err) {
    return console.error(err);
  }
  files.forEach( function (file) {
    console.log( file );
  });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to read directory /tmp
ccmzx99o.out
ccyCSbkF.out
employee.ser
hsperfdata_apache
test
test.txt
```

Remove a Directory

Syntax: Following is the syntax of the method to remove a directory –

fs.rmdir(path, callback)

Parameters

Here is the description of the parameters used −

- path − This is the directory name including path.
- callback − This is the callback function No arguments other than a possible exception are given to the completion callback.

Example

Let us create a js file named main.js having the following code –

main.js

```
var fs = require("fs");
console.log("Going to delete directory
/tmp/test");
fs.rmdir("/tmp/test",function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("Going to read directory /tmp");
  fs.readdir("/tmp/",function(err, files) {
    if (err) {
      return console.error(err);
    }
    files.forEach( function (file) {
      console.log( file );
    });
  });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
Going to read directory /tmp
ccmzx99o.out
ccyCSbkF.out
employee.ser
hsperfdata_apache
test.txt
```

Rename Files

To rename a file with the File System module, use the fs.rename() method.

The fs.rename() method renames the specified file:

Example

Rename "input2.txt" to "input3.txt":

main.js

```
var fs = require('fs');
fs.rename('input2.txt', 'input3.txt', function
(err) {
 if (err) throw err;
 console.log('File Renamed!');
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
File Renamed!
```

File Upload in node.js with express module:

- Install express module using **npm install express & npm install express-fileupload** on terminal of vs code.
- Create upload folder in your application folder.
- Create index.html and app.js file in your application folder with following code:

Index.html

```html
<h1>Hey there, Upload file here</h1>
<form method="post" enctype="multipart/form-
data" action="/">
<input type="file" name="filename">
<input type="submit" value="Upload">
</form>
```

app.js

```javascript
var express = require('express'),
   app = express(),
   http =
require("http").Server(app).listen(8081),
   upload = require("express-fileupload");
app.use(upload())

console.log("Server Started")
app.get("/",function(req,res){
  res.sendFile(__dirname+"/index.html");
})
app.post("/",function(req,res){
   if(req.files){
      var file = req.files.filename,
         filename = file.name;
      file.mv("./upload/"+filename, function
(err){
         if(err){
            console.log(err)
            res.send("error occured")
         }
         else{
            res.send("Done")
         }
      })
   }
})
```

Now run the app.js to see the result −

$ node app.js

It will upload selected file in upload folder.

Practice Programs:

1. Create a Node.js Application to count occurrence of given word in a file and display the counts on console.
2. Write Node.js application that transfer a file as an attachment in student admission form on web.
3. Create node.js application that upload image file and display image icon on browser as of your organization logo.
4. Create node.js application to accept feedback entered through a feedback form in to a file.

SET A

1. Create a Node.js file that opens the requested file and returns the content of file on terminal.
2. Using Node.js create a web page to read two file names from user and append contents of first file into second file.
3. Using Node.js create a web page to read two created files names from user and combine contents of both in to third one with all contents in Upper case.

SET B

1. Create a Node.js file that writes an HTML form, that will upload a file in particular folder
2. Create a Node.js Application to download jpg image from the Server.
3. Create a Node.js Application to check whether given name is directory or file, if it file, truncate the content after 7 bytes.

SET C

1. Create a Node.js Application to count number of lines in a given file.

Signature of the instructor: _ _ _ Date: _

Assignment Evaluation:

0: Not Done	1: Incomplete	2: Late Complete	3: Needs Improvement
4: Complete	5: Well-Done		

ASSIGNMENT NO. 3: EVENTS IN NODE.JS

Node JS follows Event Driven Single Thread Approach. Many objects in a Node emit events, for example, a net.Server emits an event each time a peer connects to it, an fs.readStream emits an event when the file is opened. All objects which emit events are the instances of events.EventEmitter.

EventEmitter Class

EventEmitter class lies in the events module. It is accessible via the following code −

```
// Import events module
        var events = require('events');
// Create an eventEmitter object
        var eventEmitter = new events.EventEmitter();
```

When an EventEmitter instance faces any error, it emits an 'error' event. When a new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired. EventEmitter provides multiple properties like **on** and **emit**. **on** property is used to bind a function with the event and **emit** is used to fire an event.

Methods : The following table lists all the important methods of EventEmitter class.

EventEmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

Events

Sr.No.	Events & Description
1	**newListener** <ul><li>**event** − String: the event name</li><li>**listener** − Function: the event handler function</li></ul>This event is emitted any time a listener is added. When this event is triggered, the listener may not yet have been added to the array of listeners for the event.
2	**removeListener** <ul><li>**event** − String The event name</li><li>**listener** − Function The event handler function</li></ul>This event is emitted any time someone removes a listener. When this event is triggered, the listener may not yet have been removed from the array of listeners for the event.

Example

Create a js file named main.js with the following Node.js code −

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
// listener #1
var listner1 = function listner1() {
  console.log('listner1 executed.');
}
// listener #2
var listner2 = function listner2() {
  console.log('listner2 executed.');
}
// Bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);
// Bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);
var eventListeners = require('events').EventEmitter.listenerCount
  (eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");
// Fire the connection event
eventEmitter.emit('connection');
// Remove the binding of listner1 function
eventEmitter.removeListener('connection', listner1);
console.log("Listner1 will not listen now.");
// Fire the connection event
eventEmitter.emit('connection');
eventListeners = require('events').EventEmitter.listenerCount(eventEmitter,'connection');
```

```
console.log(eventListeners + " Listner(s) listening to connection event");
console.log("Program Ended.");
```

Now run the main.js to see the result −

```
$ node main.js
```

Output:

```
2 Listner(s) listening to connection event
listner1 executed.
listner2 executed.
Listner1 will not listen now.
listner2 executed.
1 Listner(s) listening to connection event
Program Ended.
```

EventEmitter “emit()” function

EventEmitter class has a “emit()” function, which is used to create an Event. It takes one parameter.

eventsEmitter.emit(NameOfEventToCreate);

Here, NameOfEventToCreate: we need to pass Event Name to emit() function call as String to create that Event.

Example:-

```
var events = require("events");
var eventsEmitter = new events.EventEmitter();
eventsEmitter.emit("mobileon");
```

EventEmitter “on()” function

EventEmitter class has a “on()” function, which is used to bind an Event with an Event Handler JavaScript Function. It takes two parameters.

eventsEmitter.on(NameOfEventToBind, EventHandlerFuction);

Here, NameOfEventToBind: We need to pass Event Name a to on() function call as String to bind that event to given Event Handler JavaScript Function.

and EventHandlerFuction: Given Event Handler JavaScript Function to handle that event. It may be an anonymous JavaScript function or Plain JavaScript function.

Example:

This example is using anonymous JavaScript function as Event Handler.

```
var events = require("events");
var eventsEmitter = new events.EventEmitter();
eventsEmitter.emit("mobileon",function(data){
        console.log(data);
});
eventsEmitter.emit("mobileon");
```

We can also use Plain JavaScript function as Event Handler as shown below:

```
var events = require("events");
var eventsEmitter = new events.EventEmitter();
eventsEmitter.emit("mobileon",mobileOnHadler);
eventsEmitter.emit("mobileon");

function mobileOnHadler(data){
   console.log(data);
}
```

With this knowledge about EventEmitter class, we will develop a real-time simple example to see how Node JS handles events.

Example:

event.js

```
var EventEmitter = require('events').EventEmitter;
var myEmitter = new EventEmitter;
var customer = function(name){
// do something
   console.log('Customer Name: ' + name);
};
myEmitter.on('customer', customer);
myEmitter.on('message', function(msg){
// do something
   console.log('message: ' + msg);
});

// Execute the Application
myEmitter.emit('customer', 'Ninad');
myEmitter.emit('customer', 'Viru');
myEmitter.emit('message', 'this is the first message');
myEmitter.emit('message', 'this is the second message');
myEmitter.emit('message', 'welcome to nodejs');
```

Output:

```
Customer Name: Ninad
Customer Name: Viru
message: this is the first message
message: this is the second message
message: welcome to nodejs
```

Common Patterns for EventEmitters:

There are two common patterns that can be used to raise and bind an event using EventEmitter class in Node.js.

1. Return EventEmitter from a function
2. Extend the EventEmitter class

Return EventEmitter from a function

In this pattern, a constructor function returns an EventEmitter object, which was used to emit events inside a function. This EventEmitter object can be used to subscribe for the events. Consider the following example :

```
var emitter = require('events').EventEmitter;

function LoopProcessor(num) {
   var e = new emitter();

   setTimeout(function () {

      for (var i = 1; i <= num; i++) {
         e.emit('BeforeProcess', i);

         console.log('Processing number:' + i);

         e.emit('AfterProcess', i);
      }
   }
   , 2000)

   return e;
}
var lp = LoopProcessor(3);

lp.on('BeforeProcess', function (data) {
   console.log('About to start the process for '
+ data);
});

lp.on('AfterProcess', function (data) {
   console.log('Completed processing ' +
data);
});
```

Output:

```
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
```

In the above LoopProcessor() function, first we create an object of EventEmitter class and then use it to emit 'BeforeProcess' and 'AfterProcess' events. Finally, we return an object of EventEmitter from the function. So now, we can use the return value of LoopProcessor function to bind these events using on() or addListener() function.

Extend EventEmitter Class

In this pattern, we can extend the constructor function from EventEmitter class to emit the events.

Example:

```
var emitter = require('events').EventEmitter;
var util = require('util');
function LoopProcessor(num) {
   var me = this;

   setTimeout(function () {
       for (var i = 1; i <= num; i++) {
           me.emit('BeforeProcess', i);
           console.log('Processing number:' + i);
           me.emit('AfterProcess', i);
       }
   }
   , 2000)
   return this;
}

util.inherits(LoopProcessor, emitter)
var lp = new LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
   console.log('About to start the process for ' + data);
});
lp.on('AfterProcess', function (data) {
   console.log('Completed processing ' + data);
});
```

Output:

```
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
```

In the above example, we have extended LoopProcessor constructor function with EventEmitter class using `util.inherits()` method of utility module. So, you can use EventEmitter's methods with LoopProcessor object to handle its own events.

In this way, you can use EventEmitter class to raise and handle custom events in Node.js.

Practice Programs:

1. Write Node.js application to create an EventEmitter which will emit an event that contains information about file upload at every second.
2. Write Node.js application to read 3 file contents and display message after reading each file using event looping.
3. Create Node.js application that display message on browser when email is received in your inbox.
4. Create node.js application that change background colour of your page on button click event

SET A

1. Create Node.js Application that binds multiple custom listeners to a single event.
2. Create a Node.js event-driven application that listens multiple events, and then triggers a callback function when one of those events is detected.
3. Create Node.js application to bind custom event of receiving data from user and handles it with some listener function.

SET B

1. Create node.js application that handles mouse click event.
2. Create node.js application that handles form submission event.
3. Create node.js application that change color of text using event handling of button click.

SET C

1. Write Node.js application containing an event handler and handling event when it gets data from a file.

Signature of the instructor: _ _ _ Date: _

Assignment Evaluation:

0: Not Done	1: Incomplete	2: Late Complete	3: Needs Improvement
4: Complete	5: Well-Done		

ASSIGNMENT NO. 4: NODE.JS WITH DATABASE

Node.js can be used in database applications. One of the most popular databases is MySQL.
To be able to experiment with the code examples, you should have MySQL installed on your computer.
You can download a free MySQL database at https://dev.mysql.com/downloads/installer/

Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.
To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.
To download and install the "mysql" module, open the Command Terminal and execute the following:

```
C:\Users\Your Name>npm install mysql
```

Create Connection

Start by creating a connection to the database. Use the username and password from your MySQL database.

demo_db_connection.js

var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "*yourusername*",
  password: "*yourpassword*"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
	Save the code in a file called "demo_db_connection.js" and run the file:
	Run "demo_db_connection.js"
	C:\Users\Your Name>node demo_db_connection.js
	Which will give you this result:
	Connected!

Now you can start querying the database using SQL statements.

Query a Database

Use SQL statements to read from (or write to) a MySQL database. This is also called "to query" the database. The connection object created in the example above, has a method for querying the database:

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Result: " + result);
  });
});
```

The query method takes a sql statements as a parameter and returns the result.

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Example

Create a database named "mydb":

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb",
function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Save the code in a file called "demo_create_db.js" and run the file:
Run "demo_create_db.js"

```
C:\Users\Your Name> node
demo_create_db.js
```

Which will give you this result:

```
Connected!
Database created
```

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement. Make sure you define the name of the database when you create the connection:

Example : Create a table named "customers":

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers
(name VARCHAR(255), address
VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

Save the code in a file called "demo_create_table.js" and run the file:
Run "demo_create_table.js"

```
C:\Users\Your Name>node
demo_create_table.js
```

Which will give you this result:

```
Connected!
Table created
```

Primary Key

When creating a table, you should also create a column with a unique key for each record.
This can be done by defining a column as "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Example : Create primary key when creating the table:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (id INT
AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

Save the code in a file called "demo_create_table1.js" and run the file:
Run "demo_create_table.js"

```
C:\Users\Your Name>node
demo_create_table1.js
```

Which will give you this result:

```
Connected!
Table created
```

If the table already exists, use the ALTER TABLE keyword:

Example : Create primary key on an existing table:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT
PRIMARY KEY";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table altered");
  });
```

Insert into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

Example : Insert a record in the "customers" table:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers
(name, address) VALUES ('Company Inc',
'Highway 37')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted");
  });
});
```

Save the code above in a file called "demo_db_insert.js", and run the file:

Run "demo_db_insert.js"

```
C:\Users\Your Name>node demo_db_insert.js
```

Which will give you this result:

```
Connected!
1 record inserted
```

Insert Multiple Records

To insert more than one record, make an array containing the values, and insert a question mark in the sql, which will be replaced by the value array:

INSERT INTO customers (name, address) VALUES ?

Example

Fill the "customers" table with data:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name,
address) VALUES ?";
  var values = [
    ['John', 'Highway 71'],
```

Save the code above in a file called "demo_db_insert_multple.js", and run the file:

Run "demo_db_insert_multiple.js"

```
C:\Users\Your Name>node
demo_db_insert_multiple.js
```

Which will give you this result:

```
Connected!
Number of records inserted: 14
```

```
    ['Peter', 'Lowstreet 4'],
    ['Amy', 'Apple st 652'],
    ['Hannah', 'Mountain 21'],
    ['Michael', 'Valley 345'],
    ['Sandy', 'Ocean blvd 2'],
    ['Betty', 'Green Grass 1'],
    ['Richard', 'Sky st 331'],
    ['Susan', 'One way 98'],
    ['Vicky', 'Yellow Garden 2'],
    ['Ben', 'Park Lane 38'],
    ['William', 'Central st 954'],
    ['Chuck', 'Main Road 989'],
    ['Viola', 'Sideway 1633']
  ];
  con.query(sql, [values], function (err, result)
{
    if (err) throw err;
    console.log("Number of records inserted: "
+ result.affectedRows);
  });
});
```

The Result Object

When executing a query, a result object is returned. The result object contains information about how the query affected the table. The result object returned from the example above looks like this:

```
{
  fieldCount: 0,
  affectedRows: 14,
  insertId: 0,
  serverStatus: 2,
  warningCount: 0,
  message: '\'Records:14  Duplicated: 0  Warnings: 0',
  protocol41: true,
  changedRows: 0
}
```

The values of the properties can be displayed like this:
Return the number of affected rows:

```
console.log(result.affectedRows)
```

Which will produce this result:

```
14
```

Get Inserted ID

For tables with an auto increment id field, you can get the id of the row you just inserted by asking the result object.

Note: To be able to get the inserted id, **only one row** can be inserted.

Example : Insert a record in the "customers" table, and return the ID:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  var sql = "INSERT INTO customers (name,
address) VALUES ('Milind', 'Somatane
Village 1')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted, ID: " +
result.insertId);
  });
});
```

Save the code in a file called "demo_db_insert_id.js", and run the file:
Run "demo_db_insert_id.js"

```
C:\Users\Your Name>node demo_db_insert_id.js
```

Which will give you something like this in return:

```
1 record inserted, ID: 15
```

Selecting from a Table

To select data from a table in MySQL, use the "SELECT" statement.

Example : Select all records from the "customers" table, and display the result object:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM
customers", function (err, result,
fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

Save the code above in a file called "demo_db_select.js" and run the file: Run "demo_db_select.js"

```
C:\Users\Your Name>node demo_db_select.js
```

Which will give you this result:

```
[  { id: 1, name: 'John', address: 'Highway 71'},
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},
  { id: 3, name: 'Amy', address: 'Apple st 652'},
  { id: 4, name: 'Hannah', address: 'Mountain 21'},
  { id: 5, name: 'Michael', address: 'Valley 345'},
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},
  { id: 7, name: 'Betty', address: 'Green Grass 1'},
  { id: 8, name: 'Richard', address: 'Sky st 331'},
  { id: 9, name: 'Susan', address: 'One way 98'},
  { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},
  { id: 11, name: 'Ben', address: 'Park Lane 38'},
  { id: 12, name: 'William', address: 'Central st 954'},
  { id: 13, name: 'Chuck', address: 'Main Road 989'},
  { id: 14, name: 'Viola', address: 'Sideway 1633'}
]
```

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name.

Example : Select name and address from the "customers" table, and display the return object:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT name, address FROM customers", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

Save the code above in a file called "demo_db_select2.js" and run the file:

Run "demo_db_select2.js"

```
C:\Users\Your Name>node demo_db_select2.js
```

Which will give you this result:

```
[
  { name: 'John', address: 'Highway 71'},
  { name: 'Peter', address: 'Lowstreet 4'},
  { name: 'Amy', address: 'Apple st 652'},
  { name: 'Hannah', address: 'Mountain 21'},
  { name: 'Michael', address: 'Valley 345'},
  { name: 'Sandy', address: 'Ocean blvd 2'},
  { name: 'Betty', address: 'Green Grass 1'},
  { name: 'Richard', address: 'Sky st 331'},
  { name: 'Susan', address: 'One way 98'},
  { name: 'Vicky', address: 'Yellow Garden 2'},
  { name: 'Ben', address: 'Park Lane 38'},
  { name: 'William', address: 'Central st 954'},
  { name: 'Chuck', address: 'Main Road 989'},
  { name: 'Viola', address: 'Sideway 1633'}
]
```

The Result Object

As you can see from the result of the example above, the result object is an array containing each row as an object.

To return e.g. the address of the third record, just refer to the third array object's address property:

Example : Return the address of the third record:

```
console.log(result[2].address);
```

Which will produce this result:

```
Apple st 652
```

The Fields Object

The third parameter of the callback function is an array containing information about each field in the result.

Example

Select all records from the "customers" table, and display the *fields* object:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
```

Save the code above in a file called "demo_db_select_fields.js" and run the file:

Run "demo_db_select_fields.js"

```
C:\Users\Your Name>node
```

```
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT name, address FROM
customers", function (err, result, fields) {
    if (err) throw err;
    console.log(fields);
  });
});
```

demo_db_select_fields.js

Which will give you this result:

```
[
  {
    catalog: 'def',
    db: 'mydb',
    table: 'customers',
    orgTable: 'customers',
    name: 'name',
    orgName: 'address',
    charsetNr: 33,
    length: 765,
    type: 253,
    flags: 0,
    decimals: 0,
    default: undefined,
    zeroFill: false,
    protocol41: true
  },
  {
    catalog: 'def',
    db: 'mydb',
    table: 'customers',
    orgTable: 'customers',
    name: 'address',
    orgName: 'address',
    charsetNr: 33,
    length: 765,
    type: 253,
    flags: 0,
    decimals: 0,
    default: undefined,
    zeroFill: false,
    protocol41: true
  }
]
```

As you can see from the result of the example above, the fields object is an array containing information about each field as an object.

To return e.g. the name of the second field, just refer to the second array item's name property:

Example : Return the name of the second field:

```
console.log(fields[1].name);
```

Which will produce this result:

```
address
```

Practice Programs:

1. Create a Node.js Application that Update date of birth of given employee in “employee" table and display the result.
2. Using Node.js create Application that contains applicant details and check proper validation for (name, age, and nationality), as Name should be in upper case letters only, Age should not be less than 18 yrs and Nationality should be Indian and store the data in License database.
3. Create Node.js application that display marksheet of student on web page after accepting his roll number.
4. Create node.js application that display purchase detail of customer, after accepting orders.

SET A

1. Create a Node.js application that demonstrate create database emp DB and employee table (eid, ename, Salary) in MySQL.
2. Create a Node.js file that Select all customers from the "customers" table who purchased only mobile phones.
3. Create a Node.js application that select all customers from the "customers" table who purchased only mobile phones.

SET B

1. Create a Node.js application that finds percentage of student whose seat number is entered through input form from result table.
2. Create two tables in MySQL DB product(pcode, pname, amount) and customer(cid, cname, pcode). Find customer names who purchased television.
3. Create node js application that accepts students details through html form such as name, address , percentage, class and store it in student table.

SET C

1. Create a Node.js application that create Emp, Dept & Dept-Emp tables with 1:M relationship and display the min, max, avg salary of Employee for given department.

Signature of the instructor: _ _ _ Date: _

Assignment Evaluation:

0: Not Done	1: Incomplete	2: Late Complete	3: Needs Improvement
4: Complete	5: Well-Done		

Section-III

Advance PHP

Assignment 1: Introduction to Object Oriented Programming in PHP

Introduction:

Object-Oriented Programming (OOP) is a programming model that is based on the concept of classes and objects. As opposed to procedural programming where the focus is on writing procedures or functions that perform operations on the data, in object-oriented programming the focus is on the creations of objects which contain both data and functions together.

Object Oriented Concepts:

Before we go in detail, let’s define important terms related to Object Oriented Programming.

- **Class:** Class is a programmer-defined data type, which includes local methods and local variables. A class may contain its own constants, variables (called "properties"), and functions (called "methods").
- **Object:** An individual instance of data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Constructor:** Constructor Functions are special type of functions which are called automatically whenever an object is created. It is a special function that initializes the properties of the class.
- **Destructor:** Like a constructor function you can define a destructor function using function destruct (). You can release all the resources with-in a destructor.
- **Encapsulation:** Encapsulation means hiding or wrapping the code and data into a single unit to protect the data from outside world. It is used to protect class’s internal data (properties and method) from code outside that class and hiding details of implementation. In PHP, encapsulation is provided by visibility specifiers.
- **Inheritance:** When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Interface:** Interfaces allow you to create code which specifies which methods a class must implement, without having to define how these methods are handled. Interfaces are defined in the same way as a class, but with the *interface* keyword replacing the *class* keyword and without any of the methods having their contents defined. All methods declared in an interface must be public; this is the nature of an interface. Note that it is possible to declare a constructor in an interface.
- **Introspection:** Introspection is the ability of a program to examine an object’s characteristics, such as its name, parent class (if any), properties, and methods. With introspection, you can write code that operates on any class or object. You don’t need to know which methods or properties are defined when you write your code; instead, you can discover that information at runtime, which makes it possible for you to write generic debuggers, serializes, profilers, etc.

Implementation of Object Oriented Concepts:

Function	**Description**	**Example**

class classname [extends baseclass]	Creates a class	Class student
		{
		[var $propery [= value];…] [function functionname (arguments)
		{
		//code
		}
		….]}

$instance = new classname();	Create an object	<?php $instance1 = new myclass (); //This can also be done with a variable: $newname= 'hello'; $instance2 = new $newname(); ?>

class classname
{
function methodname()
{
Statements;
}
}
	Add a Method	<?php
		class myclass
		{ function mymethod()
		{ print " hello, myclass}}
		?>
		To invoke the method on the object $instance1, we need to invoke the operator "->" to access the newly created function mymethod
		<?php
		$instance1=new myclass();
		$instance1->mymethod();
		?>

void _construct ([mixed $args [, $. ..]])	**Constructor** is a function which is called right after a new object is created.	Method 1
		<?php
		class student
		{
		public $name;
		public $marks;
		function_construct($nm,$mk)
		{

void _destruct (void)	**Destructor** is a function which is called right after you release an object.	<?php
		class Student
		{
		var $name; var $address; var $phone;
		//This is constructor
		function _construct()
		{
		this->name="abc"; this->address="pqr"; this->phone=1111;
		}
		function_destruct()
		{
		echo "Student Object Released";}
		function printstudentinfo()
		{

		Echo this->name . ”\n”; echo this->address . ”\n”; echo this->phone . “\n”; }
		} $stud =new student(); $stud->printstudentinfo(); $stud=NULL; ?>

class extendedClass extends classname	**Inheritance**
	It is the ability of PHP to extend classes that inherit the characteristics of the parent class.
		<?php
		class myclass
		{
		//property declaration
		public $var=’a default value’;
		//method declaration
		public function desplayVar()
		{
		echo $this->var;
		}
		}
		class extendedClass extends myclass

		{
		//redefine the parent method function displayVar()
		{
		echo “Extending Class”;
		parent::displayVar();
		}
		}
		$extend =new extendedClass();
		$extend->displayVar();
		?>
		Output : Extending class a default value

class_exist()	**Introspection** We can use this function to determine whether a class exists.	$class = class_exists(classname);

get_declared_classes()	This function returns array of defined classes and checks if the class name is in returned array**.**	$classes = get_declared_classes();
get_class_methods()	We can use this function to get the methods and properties of class	$methods=get_class_methods(classna me);
get_class_vars()	This function returns only properties that have default values.	$properties=get_class_vars(classnam e);
get_parent_class()	This function is used to find the class's parent class.	$superclass=get_parent_class(classname);
is_object()	Is_object function is used to make sure that it is object.	$obj= is_obj(var);

get_class()	get_class() function is used to get the class to which an object belongs and to get class name	$classname= get_class(object);
method_exists()	This function is used to check if method on an object exists .	$method_exists=method_exists(object ,method);
get_object_vars()	This function returns an array of properties set in an object	$array=get_object_vars(object);

Interfaces	An i**nterface** is declared similar to a class but only include function prototypes (without implementation) and constants. When a class uses an interface the class must define all the methods / function of the interface otherwise the PHP engine will give you an error.
	
	The interface's function /methods cannot have the details filled in. that is left to the class that uses the interface.
		Example of an interface class duck
		{ function quack()
		{
		echo "quack,quack,qk, qk…";
		}
		}
		Interface birds
		{
		function breath(); function eat();
		}
		Class duck implements birds
		{ function quack()
		{
		echo "quack,quack,qk, qk…";
		}
		}
		function breath()
		{
		echo "duck is breathing";
		}
		function eat()
		{
		echo " duck is eating";
		}

Practice Programs:

1) Write a PHP program to create class circle having radius data member and two member functions find_circumfernce () and find_area() . Display area and Circumference depending on user's preference.
2) Create Class Collge and Class Department as base class and derived class respectively , Create one more class as Faculty to display it's detail information.(Use the concept of interface)
3) Write PHP script to demonstrate the concept of introspection for examining object.

Set A:

1) Write class declarations and member function definitions for an employee(code, name, designation). Derive emp_account(account_no, joining_date) from employee and emp_sal(basic_pay, earnings, deduction) from emp_account. Write a menu driven program
   a) To build a master table
   b) To sort all entries
   c) To search an entry
   d) Display salary
2) Define an interface which has methods area( ), volume( ). Define constant PI. Create a class cylinder which implements this interface and calculate area and volume. (Hint: Use define( ))
3) Write a Calculator class that can accept two values, then add them, subtract them, multiply them together, or divide them on request.

   For example:

   $calc = new Calculator(
   3, 4); echo $calc-
   >add(); // Displays "7"
   echo $calc- >multiply(); // Displays "12"

Set B:

1) Create a class named DISTANCE with feet and inches as data members. The class has the following member functions: convert_feet_to_inch() , convert_inch_to_feet() . Display options using radio button and display conversion on next page.
2) Write a PHP program to create a class Employee that contains data members as Emp_Name, Dept_name , Basic_sal,DA, HRA,TA , IT,PF,PT , GROSS, DEDUCTION ,NET . It has member functions calculate_gross , calculate_deductions , Calculate_net_salary . Display pay slip of employee. Create and Initialize members Emp_Name, Dept_name , Basic_sal of Employee object by using parameterized constructor.
3) Write a PHP program to create a class temperature which contains data members as Celsius and Fahrenheit . Create and Initialize all values of temperature object by using parameterized constructor . Convert Celsius to Fahrenheit and Convert Fahrenheit to Celsius using member functions. Display conversion on next page.

Set C:

1) Write a PHP program to create a class article having articleid, name, articleqty, price. Write menu driven program to perform following functions :(Use array of objects)
   i) Display details of all articles purchased.
   ii) Display details of articles whose price exceeds 500
   iii) Display details of articles whose quantity exceeds 50
2) Write a PHP program to create a class Worker that has data members as Worker_Name, No_of_Days_worked, Pay_Rate. Create and initialize the object using default constructor, Parameterized constructor. Also write necessary member function to calculate and display the salary of worker.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: well done [ ]

Signature of the Instructor

Assignment 2: To study Web Techniques

Sticky Forms:
Sticky form remembers the values you entered in the input fields. Good example of sticky form is Google search box. Sticky form helps user to type the same form again supplying the values in inputs. Sticky form is form in which the results of a query are accompanied by a search form whose default values are those of the previous query. To make sticky form , You just include the attribute value for text fields, and selected/checked for other elements:

Example :
```
<html>
<body>
<form action="<?php $_SERVER['PHP_SELF']; ?>" method="POST">
<b>Your Name : </b><input type="text" name="name" value="<?php if(isset($_POST['name'])) echo
$_POST['name'];?>">

<p><input type="submit" name="submit" value="Submit" /></p>

</form>
<?php
    echo "Your Name is =". $_POST['name']."<br>";
?>
</body>
</html>
```

Multi – Valued Parameters:

HTML selection lists, created with the select tag, can allow multiple selections. To ensure that PHP recognizes the multiple values that the browser passes to a form-processing script, you need to make the name of the field in the HTML form end with [].When PHP engine sees a submitted form field name with square brackets at the end, it creates a nested array of values within the $_GET or $_POST and $_REQUEST superglobal array, rather than a single value.

For example:

```
<select name="languages[]">
  <input name="c">C</input>
  <input name="c++">C++</input>
  <input name="php">PHP</input>
  <input name="perl">Perl</input>
</select>
```

Now, when the user submits the form , $_GET['languages'] contains an array instead of a simple string. This array contains the values that were selected by the user.

Example

```
<html>
<head><title>LANGAUGES</title></head>
<body>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
Select your Language :<br>
<select name="languages[]" multiple>
<option value="c"> C </option>
<option value ="c++"> C++ </option>
<option value ="php"> PHP </option>
<option value ="perl"> Perl </option>
</select>
<br>
<input type="submit" name="s" value="My Languages!" />
</form>
<?php
if (array_key_exists('s', $_GET))
 {
$lang = join (" ,", $_GET['languages']);
echo "You know $lang languages.";
}
?>
</body>
</html>
```

Sticky Multi – Valued Parameters:
You can make multiple selection form elements sticky. You'll need to check to see whether each possible value in the form was one of the submitted value.
For example :

RED: <input type="checkbox" name="attributes[]" value="red" <?= if (is_array($_GET['attributes']) and in_array('red', $_GET['attributes'])) { "checked"; } ?> >

Consider following example to implement sticky multi-value
parameters

```
<html>
<head><title>LANGAUGES</title></head>
<body>

<?php
$c1 = $_GET['c1'];
?>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
Qualification : <br>
<input type="checkbox" name="c1[]" value="ssc" <?php if(in_array('ssc', $_GET['c1'])) {echo
"checked"; }?>> SSC <br>

<input type="checkbox" name="c1[]" value="hsc"
<?php if(is_array($_GET['c1']) and in_array('hsc', $_GET['c1'])) { echo "checked"; }?>> HSC <br>

<input type="checkbox" name="c1[]" value="bca"
<?php if(is_array($_GET['c1']) and in_array('bca', $_GET['c1'])) { echo "checked";}?>> BCA <br>

<input type="checkbox" name="c1[]" value="mca"
<?php if(is_array($_GET['c1']) and in_array('mca', $_GET['c1'])) {echo "checked";}?>> MCA <br>

<input type="submit" name="s" value="My Qualification" />
</form>
<?php
if (array_key_exists('s', $_GET))
 {
$a = join (" ,", $_GET['c1']);
echo "You Qualification : $a";
}
?>
</body>
</html>
```

Self Processing Page:

- Self processing page means one PHP page can be used to both generate a form and process it.You can use PHP_SELF variable for generating self processing page. PHP_SELF is a variable that returns the current script being executed. This variable returns the name and path of the current file (from the root folder). You can use this variable in the action field of the FORM.
- <form name="form1" method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>" >

Example A self-processing page

```
<html>
<head><title>Temperature Conversion</title></head>
<body>

<?php
 if ($_SERVER['REQUEST_METHOD'] == 'GET') {
?>

<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
Fahrenheit temperature:
<input type="text" name="fahrenheit" /> <br />
<input type="submit" name="Convert to Celsius!" />
</form>
<?php
 }
 elseif ($_SERVER['REQUEST_METHOD'] == 'POST')
 {
   $fahr = $_POST['fahrenheit'];
   $celsius = ($fahr - 32) * 5/9;
   printf("%.2fF is %.2fC", $fahr, $celsius);
 } else
{
   die("This script only works with GET and POST requests.");
 }
?>
</body>
</html>
```

Server information :

$_SERVER is a PHP super global array which holds information about the items like Server information, Header information, Details on PHP page request, File name or path information, Remote user information, HTTP Authentication Details.

Element/Code	**Description**
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)

$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web serve
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

Example: to display server information like name , script name , user agent etc.

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

Output :

```
/php/demo_global_server.php
www.w3schools.com
www.w3schools.com
https://www.w3schools.com/php/showphp.asp?filename=demo_global_server
Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.112
Safari/537.36
/php/demo_global_server.php
```

Practice Programs:

1) Write a PHP script to Design a form to accept a number from the user to check whether number is palindrome or not. (Use the concept of self processing page)
2) Write PHP program to accept user details such as user-id, name, Address, email, and mobile no. Display same information on next page.
3) Write PHP program to create student registration form and display student information. (Use sticky form concept).

Set A:

1) Write PHP program accept name, select your cities you would like to visit and display selected information on page. (Use multi-valued parameter),.
2) Write PHP program to create student registration form and display student information. (Use sticky form concept).
3) Write a PHP script for the following: Design a form to accept a number from the user. Perform the operations and show the results.
   - Check whether number is palindrome or not.
   - Reverse the number using recursions.

   (Use the concept of self processing page.)
4) Write PHP program to select list of subjects from list box and display selected subject on information. (Use sticky multi-valued parameter)

Set B:

1) Write a PHP Script to display Server information in table format (Use $_SERVER).
2) Write a PHP program to accept two strings from user and check whether entered strings are matching or not. (Use sticky form concept).
3) Write a PHP script to accept an Indian currency and then convert it in dollar or pounds (radio buttons) according to user's preference. (use concept of self processing form).
4) Write PHP program to accept client name, property details (Flat, Bunglow, Plot), Display selected information same page. (Use multi- value parameter).

Set C:

1) Write PHP program to accept name of student , Gender(male ,female) using radio buttons ,Qualification(SSC, HSC, BCA, MCA) using check boxes . Display information of student. (Use sticky multi-valued parameter).

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: well done [ ]

Signature of the Instructor

Assignment 3 – XML

Introduction to XML:

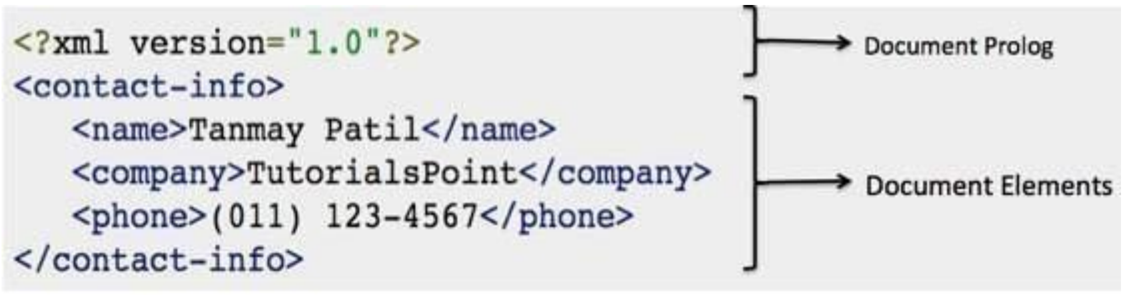
XML stands for eXtensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML). XML was designed to store and transport data. XML was designed to be both human- and machine-readable. XML is a markup language much like HTML. XML was designed to describe data. XML tags are not predefined . You must define your own tags.XML is self describing.

XML document are well – formed and valid. A well - formed XML document follows the basic XML syntax rules.A valid document also follows the rules imposed by a DTD or an XSD.

A simple document is shown in the following example −

```
<?xml version = "1.0"?>

<contact-info>

   <name>Tanmay Patil</name>

   <company>TutorialsPoint</company>

   <phone>(011) 123-4567</phone>

</contact-info>
```

The following image depicts the parts of XML document.



```
<?xml version="1.0"?>
<contact-info>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Document Prolog Section :
Document Prolog comes at the top of the document, before the root element. This section contains −

- XML declaration
- Document type declaration

Document Elements Section:
Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose.

XML declaration :
It contains details that prepare an XML processor to parse the XML document. It is optional, but when used, it must appear in the first line of the XML document.

```
<?xml version="version_number" encoding="encoding_declaration"
standalone="standalone_status" ?>
```

An XML declaration should abide with the following rules:

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case. If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.The names are always in lower case.
- The order of placing the parameters is important. The correct order is:*version, encoding and standalone.* Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

Example of XML declaration :

- <?xml >
- <?xml version="1.0">
- <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <?xml version='1.0' encoding='iso-8859-1' standalone='no' ?>

DTD :Document Type Declaration :

- The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely.
- DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.
- Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier
 [
    declaration1
    declaration2
  ........
]>
```

XML Tags :

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>. Opening and closing tags must be written with the same case.

For example,
<Message>This is incorrect</message>
<message>This is correct</message>

XML Elements :

- An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements' names are enclosed by triangular brackets < > .
- Each XML-element needs to be closed either with start or with end elements as shown below: <element>....</element>.
- An XML document can have only one root element
- An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap.
- In XML, all elements must be properly nested within each other.

XML attributes:

- An XML-element can have one or more attributes.
- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and*href* are considered two different XML attributes.
- Same attribute **cannot have two values in a syntax**

So XML follows tree structure

```
<root>
  <child>
```

```
                <subchild> .... </subchild>
               </child>
             </root>
<?xml version = “1.0” ?>
<BookStore>
     <Books>
               <PHP>
                         <title>Programming PHP</title>
                         <publication>O’RELLY</publication>
               </PHP
               >
               <PHP><title>Beginners PHP</title>
                         <publication>WROX</publication>

               </PHP
               >
     </Books>
</BookStore>
```

SimpleXML :

- SimpleXML is an extension that allows us to easily manipulate and get XML data.
- The SimpleXML extension is the tool of choice for parsing an XML document.
- SimpleXML turns an XML document into a data structure you can iterate through like a collection of arrays and objects.
- The SimpleXML extension includes interoperability with the DOM for writing XML files and built-in XPath support.
- SimpleXML is easier to code than the DOM, as its name implies.

SimpleXMLElement class represents an element in an XML document.

- To create root element of xml document, first create object of SimpleXMLElement class and initialize with root element.
- For example :
- $bk=new SimpleXMLElement("<bookstore/>");

Methods or functions of simpleXMLElement class

Function name	description	syntax	example
addChild()	The addChild() function adds a child element to the SimpleXML elemen	addChild(name,value);	$book = $bk->addchild("book");
addAttribute()	adds an attribute to the SimpleXML element.	addAttribute(name, value);	$book->addAttribute("Category" , "Technical");
getName()	Returns the name of the XML tag referenced by the SimpleXML element	getName();	$bk->getName();
asXML()	Returns a well-formed XML string (XML version 1.0) from a SimpleXML object	asXML([filename]);	echo $bk->asXML();

children()	Returns the children of a specified node as an array	children()	foreach ($book->children() as $child) { echo "Child node: " . $child . "
			"; }

attributes();	Returns the attributes/values of an element	attributes();	foreach ($book->attributes () as $k=>$v) { echo $k : $v . "
			"; }

count();	The count() function counts the children of a specified node.	count();	$cnt=$book->count();

simplexml_load_file()	Converts an XML file into a SimpleXMLEle me nt object	simplexml_load_file(file)	$xml=simplexml_load_file("note.xml");
simplexml_load_string()	The simplexml_load_str ing() function converts a well-formed XML string into a SimpleXMLElem e nt object.		<?php $note=<<<XML <note> <to>Tove</to> </note> XML; $xml=simplexml_load_string($note);

Reading XML document

```
<?php
$bk = simplexml_load_file(“book.xml”);
echo htmlspecialchars($bk->asXML());
?>
```

- With SimpleXML, all the elements in XML document are represented as tree of SimpleXMLElement objects. Any given element’s children are available as properties of elements SimpleXMLElement object.
- For example ,We can access element name as properties $book->title , $book->publisher etc.

Consider an application that reads “Book.xml” file into simple XML object. Display attributes and elements.

```
        //book .xml
<?xml version='1.0' encoding='UTF-8'?>
<bookstore>
<book category="Technical">
<title> LET US C </title>
<author> YASHWANT KANETKAR </author>
<year> 1980 </year>
</book>
<book category="Cooking">
<title> COOKING EVERYDAY </title>
<author> TARALA DALAL  </author>
<year> 2000 </year>
</book>
<book category="YOGA">
<title> LIGHT ON YOGA </title>
<author> B.K.IYENGAR </author>
<year> 1990 </year>
</book>
</bookstore>
```

```
// book.php
<?php
$xml = simplexml_load_file("book.xml");
echo $xml->getName() . "<br />";
foreach($xml->children() as $child)
  {
    echo $child->getName() . "<br>";
    foreach($child->attributes() as $k=>$v)
    {
      echo $k . "=". $v . "<br>";
      foreach($child->children() as $i=>$j)
      {
              echo $i .":". $j."<br>";

     }
   }
 }
?>
```

Practice Programs:

1) Write a XML program which shows how you can easily read and display the contents of an XML document using SimpleXML .
2) Write a script to create "Company.xml" file with multiple elements as shown below:

```
<CemployeeTeam>
    <Team Name="Red">
         <Ename>____</ Ename >
          <Eexperience>_____</ Eexperience >
          <Emobno>__</ Emobno >
          <Eaddress> _______</Eaddress>
    </Team>
   </CemployeeTeam>
```

3) Write a PHP Script to read book.XML and print book details in tabular format using simple XML.(Content of book.XML are (bookcode , bookname , author , year , price).

Set A:

1) Write a PHP script to create XML file named "Course.xml"

```
<Course>
 <SYBBA CA>
  <Student name> ......</Student name>
  <Class name>..... </Class name>
  <percentage>.... </percentage>
    </SYBBA CA>
   </Course>
```

Store the details of 5 students who are in SYBBACA.

2) Write PHP script to generate an XML code in the following format

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<CATALOG>
<CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
</CD>
</CATALOG>
```

Save the file with name "CD**.xml".**

3) Write PHP script to generate an XML code in the following format

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<CATALOG>
```

```
<PLANT>
<BOTANICAL>Sanguinaria canadensis</BOTANICAL>
<ZONE>4</ZONE>
<LIGHT>Mostly Shady</LIGHT>
<PRICE>$2.44</PRICE>
<AVAILABILITY>031599</AVAILABILITY></PLANT></CATAOG>
```

Save the file with name "plant**.xml".**

Set B:

1) Write a script to create "cricket.xml" file with multiple elements as shown below:

```
<CricketTeam>
<Team country="India">
        <player>_____</player>
         <runs>______</runs>
         <wicket>____</wicket>
 </Team>
</CricketTeam>
```

Write a script to add multiple elements in "cricket.xml" file of category, country="Australia".

2) Write a script to create "breakfast.xml" file with multiple elements as shown below:

```
<breakfast_menu>
<food>
<name>French Fries</name>
<price>Rs45</price>
<description>Young youths are very much intrested to eat it </description>
<calories>650</calories>
</food>
</breakfast_menu>
```

Write a script to add multiple elements in "breakfast.xml" file of category, Juice.

3) Create a XML file which gives details of movies available in "Mayanagari CD Store" from following categories
   a) Classical
   b) Action
   c) Horror

Elements in each category are in the following format

```
 <Category>
 <Movie Name>----</Movie Name>
 <Release Year>----</Release Year>
</Category>
```

Save the file with name "movies.xml".

Set C:

1) Create an application that reads "book.xml" file into simple XML object. Display attributes and elements (Hint:simple_xml_load_file() function) .

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: well done [ ]

Signature of the Instructor

Assignment 4: PHP with AJAX

AJAX stands for **Asynchronous JavaScript and XML**. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script. When a user wants more content, they click a link. With AJAX, a user can click something and content can be loaded into the page, using JavaScript, **without reloading the entire page.**

Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server. With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and Update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

AJAX cannot work independently. It is used in combination with other technologies to create interactive WebPages.

1) **JavaScript**
   - ✓ Loosely typed scripting language.
   - ✓ JavaScript function is called when an event occurs in a page.
   - ✓ Glue for the whole AJAX operation.
2) **DOM**
   - ✓ API for accessing and manipulating structured documents.
   - ✓ Represents the structure of XML and HTML documents.
3) **CSS**
   - ✓ Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript.
4) **XMLHttpRequest**
   - ✓ JavaScript object that performs asynchronous interaction with the server.

XMLHttpRequest is a JavaScript object capable of calling the server and capturing its response. It is used to send HTTP or HTTPS requests to a web server and load the server response data back into the script.

Creating an XMLHttpRequest Object :
All modern browsers (IE7+, Firefox, Chrome, Safari, and Opera) have a builtin XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:
xmlhttp=newXMLHttpRequest();
When a request to a server is sent, we want to perform some actions based on the response.
The onreadystatechange event is triggered every time the readyState changes. The readyState property holds the status of the XMLHttpRequest.

Three important properties of the XMLHttpRequest object:

- **readyState** :The readyState property defines the current state of the XMLHttpRequest object.

The following table provides a list of the possible values for the readyState property:

State	Description
0	The request is not initialized.
1	The request has been set up.
2	The request has been sent.
3	The request is in process
4	The request is completed.

- **OnReadyStateChange :** Determine the function called when the objects readyState changes.

```
xmlobj.onreadystatechange=function()
 {
}
```

- **responseText:**Returns the response as a string.
- **responseXML:**Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.
- **Status:**Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").
- **statusText:**Returns the status as a string (e.g., "Not Found" or "OK").
- **<u>Methods of XMLHttpRequest object</u> :**

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest objec

- **open(method, URL, async)**

Specifies the method, URL, and other optional attributes of a request. The method parameter can have a value of "GET", "POST", or "HEAD“. The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

- **send(content):** Sends the request.
- **abort()**Cancels the current request.

Practice Programs:

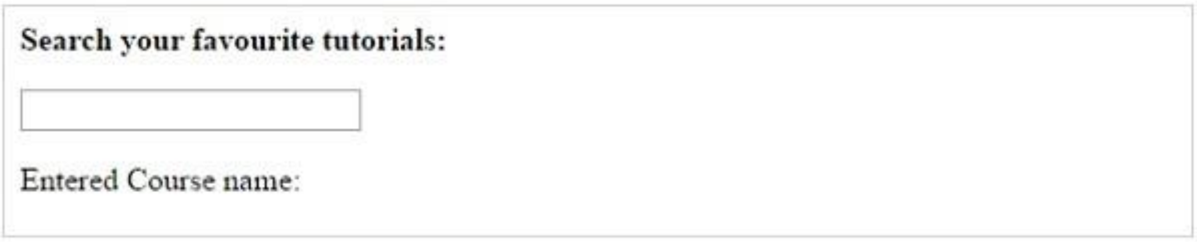
1) Write a simple PHP program which implements Ajax for addition of two numbers.
2) Write an Ajax program to display list of games stored in an array on clicking OK button.
3) Write an Ajax program to read a text file and print the contents of the file when user click on the print button.(consider “a.txt” file to create text & write text as “Ajax Example” in it.)

Set A:

1) Write a PHP script using AJAX concept, to check user name and password are valid or Invalid (use database to store user name and password).
2) Write Ajax program to carry out validation for a username entered in textbox. If the textbox is blank, print ‘Enter username’. If the number of characters is less than three,print’ Username is too short’. If value entered is appropriate the print ‘Valid username’.
3) Write Ajax program to get book details from XML file when user select a book name. Create XML file for storing details of book(title, author, year, price).

Set B:

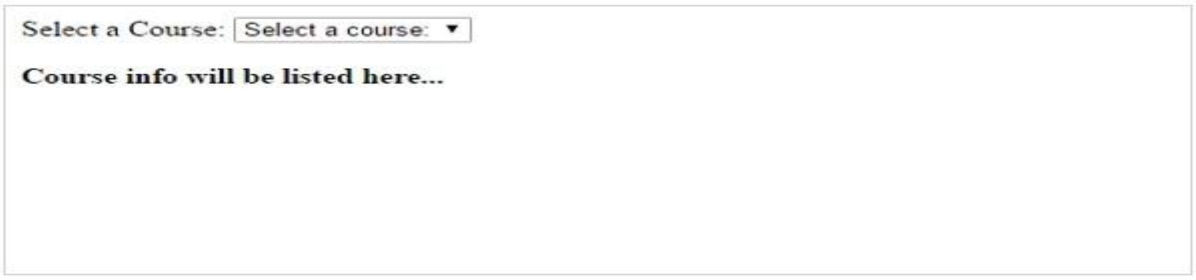
1) Write Ajax program to fetch suggestions when is user is typing in a textbox. (eg like google suggestions.Hint create array of suggestions and matching string will be displayed)
2) Write Ajax program to get player details from XML file when user select a player name. Create XML file for storing details of player (Country, player name, wickets, runs).
3) Write a AJAX program to display the following output to search your favourite tutorial from “tutorial.php” file.


Search your favourite tutorials:

Entered Course name:

Set C:

1) Write a AJAX program to display the selected course information from the list given in XML file and show the following output.


Select a Course: Select a course:

Course info will be listed here...

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: well done [ ]

Signature of the Instructor

Assignment 5: Connecting Database using PHP & AJAX

Fetch Data from MySQL table using PHP
To fetch data from the MySQL database, configure the following steps –

1) First, Include database connection file database.php
2) Assign connection variable $conn to a new variable $db
3) Create a custom function fetch_data(). This function will return data to fetch from the database. Then call fetch_data() and assign it to a new variable $fetchData.
4) Also, Create another custom function show_data($fetchData). This function returns data with an HTML table.
5) Call show_data($fetchData).
6) This function accepts a parameter to get fetched data.
File Name – backe

Function name	description	syntax	example
mysql_connect	Open a connection to a MySQL Server	mysqli_connect([$host, $username, $passwd, $dname, $port, $socket])	$conn = new mysqli($servername, $username, $password);
mysql_create_db	Create a MySQL database	mysql_create_db (string $database_name , resource $link_identifier = NULL) : bool	$sql = 'CREATE DATABASE my_db';
mysql_error	Returns the text of the error message from previous MySQL operation	mysql_error (resource $link_identifier = NULL) : string	echo mysql_errno($link) . ": " . mysql_error($link). "\n";
mysql_fetch_row	Get a result row as an enumerated array	mysql_fetch_row (resource $result) : array	$row = mysql_fetch_row($result);
mysql_db_query	Selects a database and executes a query on it	mysql_db_query (string $database , string $query , resource $link_identifier = NULL) : resource|bool	$result = mysql_query($sql, $link);
select_db()	used to change the default database for the connection.	$mysqli -> select_db(*name*)	mysqli_select_db($con, "test");
mysql_close	Close MySQL connection	mysql_close (resource $link_identifier = NULL) : bool	mysql_close($link);

Practice Programs:

1) Write an Ajax program to display list of book stored in an array on clicking ok button. (Consider Book_List.php)
2) Write an Ajax program to search the book name according the character typed & display same list using array. (Use New.php)
3) Write an Ajax program to display list of games stored in an array on clicking ok button.

Set A:

1) Write Ajax program to print Movie details by selecting an Actor's name. Create table MOVIE and ACTOR as follows with 1 : M cardinality MOVIE (mno, mname, release_yr) and ACTOR(ano, aname).
2) Create Trip table as follows
   Trip (tno, tname, Source, Destination, cost). Write Ajax program to select the trip name and print the selected trip details.
3) Create student table as follows Student(sno, sname, per).
   Write Ajax program to select the student name and print the selected student's details.

Set B:

1) Write Ajax program to get player details from player table by inserting a player name at run time display it's details in tabular form .Consider ,
   player (Country, player_name, wickets, runs).
2) Write Ajax program to calculate maximum runs scored for a particular country (Use Above Player table).

Set C:

1) Write Ajax program to get details of voters whose vage is greater than 40 year from Voter table Create voter table as Voter (vid, vname, vage, vaddress).

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: well done [ ]

Signature of the Instructor

Assignment 5: PHP Framework - Druple

Drupal is open source software that allows publishing, managing and organizing a wide variety of content on a website easier. Drupal is used to easily manage, update and publish the content in the website. Many individuals and organizations are using Drupal to create professional websites to suit their custom requirements. Because of easy creating sites, application and management, Drupal is used by many organizations. We can enhance the functionality of Drupal by adding available add-on modules.

Creating Contents

You can add two types of contents in your website: Article and Basic Page.
To create content click the link "Add content." From the short cut menu.
Choose between Article and Basic page.

Creating Articles

Content type Article has the following features:

- Summary posted to the front page of the Web site.
- Comments enabled.
- An image can be displayed with the article.
- User name of the article author as well as the time it was originally published.
- Tags enabled, allowing you to categorize articles.

To create an article, do the following steps:

- From the shortcut menu, click the link "Add content." An overlay will appear prompting you to choose between Article and Basic page.
- Click "Article".
- Enter a title and body for your page.
- Scroll to the bottom and click Save.

Creating Basic Page

Content type Basic Page has the following features:

- Are not published to the front page of your Web site.
- Do not allow visitors to post comments.
- Do not have tagging enabled.
- Do not have an image upload widget.
- Are not date-stamped.

To create a Basic Page, do the following steps:

- From the shortcut menu, click the link "Add content." An overlay will appear prompting you to choose between Article and Basic page.
- Click "Basic page".
- Enter a title and body for your page.
- Scroll to the bottom and click Save.

Customizing the Display

Use the following steps to change the theme and logo image of your website:

- Using the administrative dashboard, click the tab Appearance.
- Scroll down to the bottom of the screen (where all the disabled themes live), and beneath your theme's screen shot, click the link Enable and set default."

After the screen refreshes, click the settings link for your theme.

- Scroll to the fieldset "Logo image settings." Unselect the check box "Use the default logo." A new set of settings will be revealed.
- Click Browse and find your logo image for this theme on your hard drive.
- Scroll to the bottom of the screen and click "Save configuration."

Blocks

- Blocks can be placed into any region in your theme.

To create Block, do the following steps:

- Using the administrative dashboard, navigate to Structure > Blocks.
- Click the link “Add block”.
- Enter description and the text.
- Scroll to the bottom and click “Save block.”

Modules

The modules are used to create, edit, and delete content; convert URLs into specific database requests to retrieve content; and create the menus you use to navigate your Web site.
Modules are little programs that allow you to do more things with your Website. Modules are set of files contained in a Drupal folder. These files may include the following:

- An information file that describes the module to Drupal. This file lists the version, files within the module directory, configuration screen shots, and a short description of the module. This file is required.
- Installation instructions for Drupal that create the necessary database tables for the module. This file is required.
- PHP scripts that hook into Drupal and allow you to perform specific tasks.
- Template files responsible for the output of the module. These template files can be altered by your theme. These files are optional.
- CSS files, JavaScript files, and images. These files are optional.

Practice Programs:

1) Create a Page in Drupal titles “Game”. Add the details of different games (football, hockey, and cricket) with player list on the page. The page should contain announcement about upcoming match.
2) Create a module in Drupal To design a form with the following components:
   Item - Ino, IName, and Rate
   One submit button.

   After submitting the form insert a Item record into a table named Item. Also display a message when the record is inserted successfully, and fetch the Item from the table and display “<Iname>=<Rate>. Also add Navigation on the Home Page called “Item Rate”.
3) Using Drupal create a module containing details of your college. On the home page add Navigation which contains your college name and also add logo image.

Set A:

1) Create a Basic Page in Drupal titled “About Me”. Add the details about yourself in the page. Also place this page link in the Main Menu. Display this menu link before all the menu items. Show text “This is <your name>” when move the mouse pointer at this menu link.
2) Develop a module in Drupal to create a page showing your contact details (name, roll_no, address, phone). Also add Navigation on the Home Page called “Contact Details”.
3) Using Drupal create a page showing the teacher details (name, contactno, subjecttaught). Add Navigation on the home page called “Teacher Details”

Set B:

1) Create a Block in Drupal titled “Event”. The block should be displayed in the left side of each page. The block should contain announcement about an upcoming events. Also change the theme of your website by following properties:
   a) Change the background colour.
   b) Change the logo image.
2) Create a front-page article in Drupal titled “My Article”. Write an article about PHP programming Language and add to the article page. Display an Image appropriate to the Article at the bottom of the Article. Also place this page link in the Main Menu. Display this menu link before all the menu items. Show text “This is <your name>” when move the mouse pointer at this menu link. Also post a comment about.
3) Create a module in Drupal To design a form with the following components:

Text Fields - Roll No, Name, and Address
One submit button.

After submitting the form insert a student record into a table named student. Also display a message when the record is inserted successfully, and fetch the name of student from the table and display "Hello: <student name>". Also add Navigation on the Home Page called "Student Form".

Set C:

1) Develop a module in Drupal to design a registration form with the following fields:
   Text Field – First Name, Last Name, email, city List Boxes – Select Country, Date of Birth (Separate Select Boxes for month, day, and year)
   Radio Buttons – Gender - Male/Female
   Check Boxes – Technology Known – Java, PHP
   One Browse button to upload picture.
   Perform validation to check if the First Name and Last Name are not empty and the email is valid. If that is not the case display error message and the form will not be submitted.
   Display message "Form has been submitted successfully" after clicking on the Submit button.
   Also add a Navigation on the Home Page called "Registration".

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete []
3: Needs Improvement [ ]	4: Complete [ ]	5: well done [ ]

Signature of the Instructor