Table of Contents

Introduction

About the workbook

This workbook is intended to be used by S. Y. B. Sc (Computer Science) students for the Object-Oriented Programming using C++ Lab course Semester IV.

Object Oriented Programming using C++ is an important core subject of computer science curriculum, and hands-on laboratory experience is critical to the understanding of theoretical concepts studied as part of this course. Study of any programming language is incomplete without hands on experience of implementing solutions using programming paradigms and verifying them in the lab. This workbook provides rich set of problems covering the basic algorithms as well as numerous computing problems demonstrating the applicability and importance of various data structures and related algorithms.

The objectives of this book are

- Defining clearly the scope of the course
- Bringing uniformity in the way the course is conducted across different colleges
- Continuous assessment of the course
- Bring variation and variety in experiments carried out by different students in a batch
- Providing ready reference for students while working in the lab
- Catering to the need of slow paced as well as fast paced learners

How to use this workbook

The Object-Oriented Programming using C++ syllabus is divided into five assignments. Each assignment has problems divided into three sets A, B and C.

- Set A is used for implementing the basic algorithms or implementing data structure along with its basic operations. **Set A is mandatory.**
- Set B is used to demonstrate small variations on the implementations carried out in set A to improve its applicability. Depending on the time availability the students should be encouraged to complete set B.

- Set C prepares the students for the viva in the subject. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

Instructions to the students

Please read the following instructions carefully and follow them.

- Students are expected to carry workbook during every practical.
- Students should prepare oneself before hand for the Assignment by reading the relevant material.
- 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 as many problems as possible given in this work book.
- Students will be assessed for each exercise on a scale from 0 to 5
  - Not done 0
  - Incomplete 1
  - Late Complete 2
  - Needs improvement 3
  - Complete 4
  - Well Done 5

Instruction to the Practical In-Charge

- Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.
- Choose appropriate problems to be solved by students. Set A is mandatory. Choose problems from set B depending on time availability. Discuss set C with students and encourage them to solve the problems by spending additional time in lab or at home.
- Make sure that students follow the instruction as given above.
- You should 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 page of the respective Lab course.

Instructions to the Lab administrator and Exam guidelines

- You have to ensure appropriate hardware and software is made available to each student.
- Do not provide Internet facility in Computer Lab while examination
- Do not provide pen drive facility in Computer Lab while examination.

The operating system and software requirements are as given below:

- Operating system: Linux
- Editor: Any Linux based editor like vi, gedit etc.
- Compiler: g++

Assignment Completion Sheet

Lab Course I			
Object Oriented Programming using C++			
Assignment Number	**Assignment Name**	**Marks (out of 5)**	**Teachers Sign**
1	Class and Objects, Array of Objects		
2	Functions a. Inline function b. static function		
3	Overloading a. Function Overloading b. Operator Overloading		
4	Inheritance, Function overriding		
5	Virtual Functions, Pure virtual function		
Total (Out of)			
Total (Out of)			
Total			

This is to certify that Mr/Ms _____

University Exam Seat Number _____ has successfully completed the course work for Lab Course I and has scored _____ Marks out of 15.

Instructor **Head**

Internal Examiner **External Examiner**

Section I

Object Oriented Programming using C+

Assignment 1: Class and Object, Array of Objects

Reading

You should read following topics before starting this exercise:

1. Difference between C and C++.
2. Features of C++
3. Concept of Translators, Compilers
4. C++ functions

Ready References

1.1 C++ Keywords

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

1.2 Data Types

Basic Data Types

Type	Bit Width	Range
Char	8	-128 to +127
wchar_t	16	0 to 65,535
int (16-bit environments)	16	-32,768 to 32,767
int (32-bit environments)	32	-2,147,483,648 to +2,147,483,647
Float	32	3.4E-38 to 3.4E+38
Double	64	1.7E-308 to 1.7E+308
Bool	N/A	true/false
Void	N/A	valueless

Data Types with Modifiers in Combination

Type	Bit Width	Range
unsigned char	8	0 to 255
signed char	8	-128 to +127
unsigned int	32	0 to 4,294,967,295
signed int	32	-2,147,483,648 to 2,147,483,647
short int	16	-32,768 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	-32,768 to 32,767
long int	32	same as int
unsigned long int	32	same as unsigned int
signed long int	32	same as signed int
long double	80	3.4E-4932 to 1.1E+4932

1.3 Operators In C++

Operator	Description	Example	Overloadable
Group 1 (no associativity)			
::	Scope resolution operator	Class::age = 2;	NO
Group 2			
()	Function call	isdigit('1')	YES
()	Member initalization	c_tor(int x, int y) : _x(x), _y(y*10){};	YES
[]	Array access	array[4] = 2;	YES
->	Member access from a pointer	ptr->age = 34;	YES
.	Member access from an object	obj.age = 34;	NO
++	Post-increment	for(int i = 0; i < 10; i++) cout << i;	YES
--	Post-decrement	for(int i = 10; i > 0; i--) cout << i;	YES
const_cast	Special cast	const_cast<type_to>(type_from);	NO
dynamic_cast	Special cast	dynamic_cast<type_to>(type_from);	NO
static_cast	Special cast	static_cast<type_to>(type_from);	NO
reinterpret_cast	Special cast	reinterpret_cast<type_to>(type_from);	NO
typeid	Runtime type information	cout « typeid(var).name(); cout « typeid(type).name();	NO
Group 3 (right-to-left associativity)			
!	Logical negation	if(!done) …	YES
not	Alternate spelling for !		
~	Bitwise complement	flags = ~flags;	YES
compl	Alternate spelling for ~		

++	Pre-increment	for(i = 0; i < 10; ++i) cout << i;	YES
--	Pre-decrement	for(i = 10; i > 0; --i) cout << i;	YES
-	Unary minus	int i = -1;	YES
+	Unary plus	int i = +1;	YES
*	Dereference	int data = *intPtr;	YES
&	Address of	int *intPtr = &data;	YES
new	Dynamic memory allocation	long *pVar = new long; MyClass *ptr = new MyClass(args);	YES
new []	Dynamic memory allocation of array	long *array = new long[n];	YES
delete	Deallocating the memory	delete pVar;	YES
delete []	Deallocating the memory of array	delete [] array;	YES
(type)	Cast to a given type	int i = (int) floatNum;	YES
sizeof	Return size of an objector type	int size = sizeof floatNum;int size = sizeof(float);	NO
Group 4			
->*	Member pointerselector	ptr->*var = 24;	YES
.*	Member object selector	obj.*var = 24;	NO
Group 5			
*	Multiplication	int i = 2 * 4;	YES
/	Division	float f = 10.0 / 3.0;	YES
%	Modulus	int rem = 4 % 3;	YES
Group 6			
+	Addition	int i = 2 + 3;	YES
-	Subtraction	int i = 5 - 1;	YES
Group 7			
<<	Bitwise shift left	int flags = 33 << 1;	YES
>>	Bitwise shift right	int flags = 33 >> 1;	YES
Group 8			
<	Comparison less-than	if(i < 42) …	YES
<=	Comparison less-than-or-equal-to	if(i <= 42) ...	YES
>	Comparison greater-than	if(i > 42) …	YES
>=	Comparison greater-than-or-equal-to	if(i >= 42) ...	YES
Group 9			
==	Comparison equal-to	if(i == 42) ...	YES
Eq	Alternate spelling for ==		
!=	Comparison not-equal-to	if(i != 42) …	YES
not_eq	Alternate spelling for !=		
Group 10			

&	Bitwise AND	flags = flags & 42;	YES
bitand	Alternate spelling for &		
Group 11			
^	Bitwise exclusive OR (XOR)	flags = flags ^ 42;	YES
xor	Alternate spelling for ^		
Group 12			
|	Bitwise inclusive (normal) OR	flags = flags | 42;	YES
bitor	Alternate spelling for |		
Group 13			
&&	Logical AND	if(conditionA && conditionB) …	YES
and	Alternate spelling for &&		
Group 14			
||	Logical OR	if(conditionA || conditionB) ...	YES
Or	Alternate spelling for ||		
Group 15 (right-to-left associativity)			
? :	Ternary conditional(if-then-else)	int i = (a > b) ? a : b;	NO
Group 16 (right-to-left associativity)			
=	Assignmentoperator	int a = b;	YES
+=	Increment andassign	a += 3;	YES
-=	Decrement andassign	b -= 4;	YES
*=	Multiply and assign	a *= 5;	YES
/=	Divide and assign	a /= 2;	YES
%=	Modulo and assign	a %= 3;	YES
&=	Bitwise AND andassign	flags &= new_flags;	YES
and_eq	Alternate spelling for &=		
^=	Bitwise exclusive or (XOR) and assign	flags ^= new_flags;	YES
xor_eq	Alternate spelling for ^=		
|=	Bitwise normal ORand assign	flags |= new_flags;	YES
or_eq	Alternate spelling for |=		
<<=	Bitwise shift left and assign	flags <<= 2;	YES
>>=	Bitwise shift rightand assign	flags >>= 2;	YES
Group 17			
throw	throw exception	throw EClass(“Message”);	NO
Group 18			
,	Sequential evaluation operator	for(i = 0, j = 0; i < 10; i++, j++) …	YES

1.4 C++ Program Structure

Example 1: Simple Program in C++ to calculate weekly pay.

```
#include <iostream>

using namespace std;int
main ()
{
    int workDays;
    float  workHours, payRate, weeklyPay;

    workDays = 5;
    workHours = 7.5;
    payRate = 38.55;
    weeklyPay = workDays  * workHours *
    payRate;cout << "Weekly Pay = ";
    cout <<weeklyPay;
    cout << '\n';

}
```

1. The statement **#include <iostream>** is a preprocessor directive that includes the necessary definitions so that a program can do input and output using the iostream library.
2. The statement **using namespace std**
3. The main() function can have zero or more parameters. The function may have return type as int or void. All C++ programs must have exactly one main function. Program execution always begins from main.
4. The brace marks the beginning of the body of main.
5. The line defines an integer variable called as **workdays**.
6. The line defines three float variables – **workHours**, **payRate**, **weeklyPay**
7. The line is an assignment statement. It assigns value 5 to the variable **workDays**.
8. The line assigns 7.5 to the variable **workHours**.
9. The line assigns 38.55 to the variable **payRate**.
10. The line calculates **weeklyPay** as a product of **workdays**, **workHours** and **payRate**.

11-13 The symbol << is an output operator which takes an output stream as its left operand and an expression as its right operand.

These lines will produce following output:

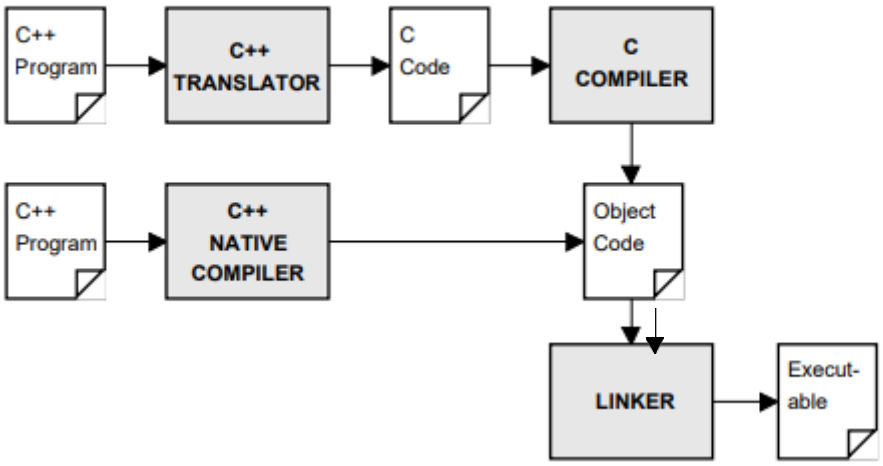
"Weekly Pay = 1445.625"

14. This brace marks the end of the body of main.

Compiling a C++ Program

```
$ g++ pay.cpp
$ ./a.out
Weekly Pay = 1445.625
$
```

C++ Compilation Process



Example 2: A simple C++ program to compute average velocity of car.

```
int main()
{
 cout << "Start Milepost?";
 int StartMilePost;
 cin >> StartMilePost;
 cout << "End Milepost?";
 int EndMilePost;
 cin >> EndMilePost;
 cout << "End time (hours, minutes, seconds)?";
 int EndHour, EndMinute, EndSecond;
 cin >> EndHour>>EndMinute>>EndSecond;
 float ElapsedTime = EndHour + (EndMinute/60.0)+(EndSecond /
3600.0);
 int Distance = EndMilePost - StartMilePost;
 float Velocity = Distance / ElapsedTime;

 cout << "\nCar traveled " << Distance << " miles in ";
cout << EndHour << " hrs " << EndMinute << " min "
<< EndSecond << " sec\n";
 cout << "Average velocity was " << Velocity << " mph" <<
endl;
 return 0;
}
```

```
$g++ velocity.cpp
$./a.out
Start Milepost?100

End Milepost?1000

End time (hours, minutes, seconds)?12
```

a. Type Conversions in C++

Expression Evaluation is the process of applying the operation to the operands. A set of conversions are applied to operands before binary operations are applied. These conversionsare called as the *unary binary conversions.*

Result types for integer binary operations

Type of left operand	Type of right operand			
	char	**char**	**int**	**long**
char	int	int	int	long
short	int	int	int	long
int	int	int	int	long
long	long	long	long	long

Result types for binary floating point operations

Type of left operand	Type of right operand		
	float	**double**	**long double**
Float	float	double	long double
double	double	double	long double
long double	long double	long double	long double

Result types for mixed mode arithmetic operations

Type of left operand	Type of right operand				
	int	**long**	**float**	**double**	**long double**
int	int	long	float	double	long double
long	long	long	float	double	long double
float	float	float	float	double	long double
double	double	double	double	double	long double
long double	long double	long double	long double	long double	long double

1.6 Casts

It is possible to force an expression to be of a specific type by using a construct called cast. The general form of this cast is

(type)expression

For example, if you want to make sure the expression (x/2) is evaluated to type float, you can write

(float) x / 2 Example 3: Use of cast in C++.

```
1  #include <iostream> using namespace std;int main()
2  {
3      int i; for(i=1;i<100;i++)
4
5       cout << i << "/  2 "<< (float) i / 2 << '\n' ;
6      return 0;
7  }
```

Compile above program using g++ and run.

NOTE:

- *C++ defines five casting operators. The first is the traditional – style cast inherited from C. The remaining four casting operators are* dynamic_cast, const_cast, reinterpret_cast, *and* static_cast.
- *Another feature of C++ is* Run- Time Type Identification (RTTI).

Class: -

It is a user defined data type which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
For example: - Consider class of students.

There may be many students with different names and Qualities but all of them will share some common properties like all of them will have Roll No, Name, Class, Branch etc. So here, student is the class and rollno, name, class, branch is their properties.

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.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end

Class Classname

{

access specifier: // it can be public, private or protected

Data member; // Variables of the class defined

Member functions ( ) {}; // Methods to access data members

}; // Class ends with semicolon

For the above class student, class will be defined as follows:

```
class student
{
 public:
int rollno;
char name[20]; char class[20];
char branch[20];     //Data Members
void getdata();      //Member Function
void putdata();      // Member Function
};
```

A class can have public, private or protected label sections for access specifiers. The default access for data members is private.
Public: - public member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function.
Private: -A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.
Protected: - It is member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

Defining Member Functions:-
Member functions are the functions, which have their declaration inside the class definition and works on the data members of that class. The definition of member functions can be inside or outside the definition of class. If the member function is defined inside the class definition it can be defined directly, but if its defined outside the class, then we have to use the scope resolution operator along with class name along with function name.
Member function defined inside the class:-
Member function inside the class does not require to be declared first here we can directly define the function.

```
class student
{
int rollno;
char name[20]; char class[20];
char branch[20];     //Data Members
void getdata()       //Member Function defined inside the class
{
cout<<"enter rollno"; cin>>rollno; cout<<"enter name"; cin>>name; cout<<"enter branch";
cin>>branch;
}
};
```

Member function defined outside the class:-
In this case we must first declare the function inside the class and then define the function outside the class using scope resolution operator.

```
class student
{
public:
int rollno;
char name[20];
 char class[20];
```

```
char branch[20];     //Data Members
void getdata();
 void putdata();
};
void student::getdata()       // Use of scope resolution operator
{
cout<<"enter rollno"; cin>>rollno; cout<<"enter name"; cin>>name; cout<<"enter branch";
cin>>branch;
}
```

Array of Objects: -

Like array of other user-defined data types, an array of type class can also be created. The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type.

The syntax for declaring an array of objects is

class_name array_name [size];

Simple example illustrating use of array of objects.

```
#include<iostream.h> class books
{
char tit1e [30]; float price ;
 public:
voidgetdata ( );
void putdata ( );
};
void books :: getdata ( )
{
cout<<"Title:"; cin>>title; cout<<"Price:"; cin>>price;
}
void books :: putdata ( )
{
cout<<"Title:"<<title<<"\n"; cout<<"Price:"<<price<< "\n";
}
int main ( )
{
books book[4] ;      // Array of books created
for(int i=0;i<4;i++)
{
cout<<"Enter details o£ book "<<(i+1)<<"\n"; book[i].getdata( );
}
for(int i=0;i<4;i++)
{
cout<<"\nBook "<<(i+l)<<"\n"; book[i].putdata( ) ;
}
return 0;
}
```

Set A

1. Write the definition for a class called Cylinder that contains data member's radius and height. The class hasthe following member functions:
   a. void setradius(float) to set the radius of data member
   b. void setheight(float) to set the height of data member
   c. float volume( ) to calculate and return the volume of the cylinder

d. float area( ) to calculate and return the area of the cylinder.
Write a C++ program to create two cylinder objects and display each cylinder and its area and volume.

2. Create a class named DISTANCE‟ with: - feet and inches as data members. The class has the followingmember functions:
   a. To input distance
   b. To output distance
   c. To add two distance objects
   Write a C++ program to create objects of DISTANCE class. Input two distances and output the sum.

3. Write the definition for a class called ‘time’ that has hours, minutes & seconds as integer data members.
   The class has the following member functions:
   void settime(int, int, int) to set the specified values of hours, minutes and seconds in object
   void showtime() to display contents of time object time
   time add(time) add the corresponding values of hours, minutes and seconds in time object argument to current time object and make appropriate conversions and return time
   time diff(time) subtract values of hours, minutes and seconds in time object argument from current time object after making appropriate conversions and return time difference

Set B

1. Define a class account with following specifications
   private data members account number – automatically generated six digit account number, first two digit are used for bank code (assume the value 82) and next four digits for account number account type – it can be one of the following type { saving, current, fixed, recurring} amount – long integer for the balance amount
   Owner Name – name of the owner
   Public data members for
   Setting and getting account type, initial amount and Owner Name
   Display account information
   Write a main program to accept information from user and open at least five accounts and display their information.
2. Implement a class ‘RomanNumeral’ which stores the RomanNumeral as a string. Write a member function getDecimal which returns the decimal value of the Roman numeral. The decimal values of roman numerals are as follows M 1000, D 500, C 100, L 50, X 10, V 5 and I 1. Write the program and create the roman numerals such as VIII, MCXIV, MCDLXVI and print their decimal values.

Set C

1. Write a C++ program to create a class employee having Emp_no, Name, Basic, DA, HRA, Allowances. Write necessary member functions to accept and display details of Employee and generate a Pay slip using appropriate manipulators for formatted display.
2. Having declared two variables t1 and t2 of type time, what happens if we write the following statement? Justify. t1=t2;

Assignment Evaluation

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

Practical In-charge

Assignment 2: Functions

a. Inline function

Functions are the building blocks of C++ programs where all the program activity occurs. Function is a collection of declarations and statements.

A large single list of instructions becomes difficult to understand. For this reason functions are used. A function has a clearly defined objective (purpose) and a clearly defined interface with other functions in the program. Reduction in program size is another reason for using functions.

Inline Function:

Inline function is one of the important features of C++.

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 incase defined function is more than a line.

Syntax

```
Inline return type function-name(argumentlist)
{
//FunctionBody
}
```

Example of inline function to return max of two numbers−

```
#include<iostream.h>
inline int Max(int x)
{
return(x >y)?x : y;
}
int main()
{
cout<< "Max (20,10): " << Max(20,10) <<endl; cout<< "Max (0,200): " << Max(0,200)
<<endl; cout<<"Max(100,1010):"<<Max(100,1010) <<endl;
return 0;
}
```

Static Variables

You already know that each object has its own set of member variables. However, in some situations, it is preferable to have one or more variables that are common for all objects. C++ provides static variable for this purpose.

As its name suggests, static variables retain their value even after the function to which it belongs has been executed. This means that a static variable retains its value throughout the program.

NOTE:

The static variables have to be initialized explicitly either inside the class declaration or outside the class.

Static variables are initialized outside the member functions or class definition. The following example illustrates the declaration and initialization of a static variable.

EXAMPLE:

```
Class staticExample
 {
        Int data;
        static int staticVar;//Static variable declared public:
          //Defination of member function
 };
 Int staticExample::staticVar=0;//Staticvariableinitialized to 0
```

In the above example, static variable is initialized outside the class definition. Creating more than one object will not re-initialize the static variable. Unlike member variables, only one copy of static variable exists in memory for all objects of that class. Thus, all objects share one copy of the static variable in memory. Consider a situation where you are required to keep track of the number of objects created for a given class. If you declare a non-static member variable called counter, every instance of that class will create as separate variable, and there would be no single variable having a count of the total number of objects created. To avoid this, you use static member variables because it is this single static variable that gets incremented by all objects.

You can also initialize the static variable inside the class definition as shown in the following example.

Example 2:

```
#include<iostream>

class myclass
{
  Void increment()
  {
    Static int
    i=5;
    cout<<i<<endl;
  }

  public:
  void display()
  {
    cout<<"callingincrementforfirsttime"<<endl;
    increment();
    cout<<"callingincrementforsecondtime"<<endl;
    increment();
  }
};

Int main()
{
   myclass m1;
   m1.display();
   return0;
}
```

Compile and run this program to see the output.

Static Functions

Like static variables, you can also declare a function to be static. Static functions can only access static variables and not non-static variables. You use the class name to access the static function, as shown below:

ClassName::StaticFunctionName;

The following code samples illustrates the use of static functions:

In a situation where you want to check whether or not any object of a class has been created,

```
Class staticExample
{
        Int data;
        Static int staticVar;//Static  variable declared public:
                Static void display()
                {
                        cout<<"staticVar="<<staticVar";
                        cout<<"data="<<data;//Error!Staticfunctions
                                                //cannotaccessnon-
                                                //staticvariables.

                }
};
int staticExample :: staticVar=0;// Static variable initialized to 0
 void main()
 {
   staticExample::display();//Withoutcreatinganobjectofaclass,the
                                //staticfunctioncanbeaccessed
 }
```

Compile and run this program to see the output.

you make use of static functions because they come into existence even before the object is created.

In a situation where you want to check whether or not any object of a class has been created, you make use of static functions because they come into existence even before the object is created.

Set A

1. Write a C++ program to print area of circle, square and rectangle using inline function.
2. Write a C++ program to subtract two integer numbers of two different classes using friend function.
3. Write a C++ program to calculate function to determine simple interest by using default arguments as follows int calculate(intp,int n=10,int r=7) Returns SI by specifying no of years and rate of interest
4. Write a C++ program with class operations have two data members num1 and num2
   Define following inline function
   add() that adds two variable
   sub() to print difference
   mul() to multiply two numbers
   div() to divide two numbers

5. create a class check which have the method ispalindrome(string) which check whetheher a word is palindrome or not a

Set B

1. Write a C++ program to accept records of n employees and store it in array. Use array of objects. The class of employee should be as follows:
   class Employee
   {
   int EmpNo;
   char Name[20];
   char Gender;
   long Salary;
   };
   Implement following search functions:
   Employee searchNum(Employee arr[], int EmpNo);
   Employee searrchname(Employee arr[], char Name[]);
   Employee searchgender(Employee arr[], char Gender);
   Employee searchSal(Employee arr[], long Salary);
   Also implement a function to return an employee who is drawing highest salary by using following function:
   Inline Employee compare(long a, long b);

2. Write a program to encrpt the string provided by user (ask the user to shift the number of characters)(for e.g "abc" will be encrypted to "def"-3 shift (3- letters down)

3. Write a program to count the number of students :having data member name,age and marks (use static variable)

4. Write a program to accept 'n ' numbers from user, using concept of array of objects and find the count of Prime number(use the concept of static function)

Set C:

1. Define a class ‘Fraction’ having integer data members numerator and denominator. Define parameterized and default constructors (default values 0 and 1). Parameterized constructor should store the fraction in reduced form after dividing both numerator and denominator by gcd(greatest common divisor). Write a private function member to compute gcd of two integers.

Assignment Evaluation

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

Practical In-charge

Assignment 3: Overloading

Function Overloading

Function overloading is a kind of polymorphism in which there can be several functions with same name but different set of parameters, all functions conceptually carrying out the same task but there is variation depending on the arguments. The definition of functions should differ from each other by type and /or number of arguments in the argument list. One commonly found overloaded function in a Class is the constructor itself as there can be more than one possible ways of constructing an object.

Set A

1. Implement a class 'printdata' with three member functions all with the same name 'print' void print(int) - outputs value - <int>, that is, value followed by the value of the integer void print (int, int) – outputs value – [<int>, <int>], that is, value followed by the two integers separated by comma in square brackets.void print(char *) – outputs value – "char*", that is, value followed by the string in double quotes.
   Write a main function that uses the above class and its member functions.

2. Write a C++ program to overload function volume and find volume of cube, cylinder and sphere(use concept of class and objects)

3. Implement a class 'maxdata' with two member functions both with the same name 'maximum' int maximum (int, int) – returns the maximum between the two integer arguments int maximum (int *) – returns the maximum integer in the array of integers
   Write a main function that uses the above class and its member functions.

Set B

1. Implement a class 'invertdata' with three member functions all with the same name 'invert' int invert (int) - returns the inverted integer – invert(5438) will return 8345
   char * invert (char *) – returns the reversed string – reverse("comp") will return "pmoc"
   void invert(int *) – will reverse the array order – An array [5, 7, 12, 4] will be inverted to [4, 12, 7, 5]

Set C

1. Can we have two functions with same name and set of arguments but different return data types?
2. Can we have two functions with same name with two different data types, for one function the argument is int for other it is user defined type 'Number' which is basically int but renamed 'Number' using typedef?
3. Can we have two functions void f(int x) and void f(int & x), that is, one with integer argument and other with reference to integer argument?

Assignment Evaluation

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

Practical In-charge

Assignment 3B: Operator overloading.

Operator overloading gives special meaning to usual operators like +, *, < etc. for user defined types similar to the one they have with built in data types.
There are few operators
in C++ that cannot be overloaded such as ternary operator ?:, sizeof, scope resolution operator :: and membership operators . and .* .
Operator overloading is carried out by writing member functions using the operator keyword called operator functions.

The general syntax of an operator function is

```
return-type operator operatorsymbol ( parameter list)
{
}
```

Here return-type is commonly the name of the class itself as the operations would commonly return object of that class type.
The argument list will depend on whether the operator is unary or binary and whether the function is a member function or friend function. For example for a unary operator, member function will have no arguments as the class object itself is the object on which operator operates. For a binary operator, a non-member function will have two arguments while a member function has one argument, the other implicitly being the class object itself.

```
fraction operator + ( fraction f)
{
fraction temp;
temp.numerator = numerator*f.denominator +denominator*f.numerator
temp. denominator = denominator*f.denominator;
temp.numerator = temp.numerator/gcd(temp.numerator, temp.denominator);
temp.denominator = temp.denominator/ gcd(temp.numerator,temp.denominator);
return temp;
}
```

Overloading increment and decrement operators

The increment operator ++ has two forms: pre-increment (++u) and post-increment(u++). To distinguish between pre and post increment operator overloading, we use a dummy parameter of type int in the function heading of the post-increment operator function. Decrement operator can be overloaded similarly.

```
fraction operator++()
{
numerator+=denominator;// add one to the fraction, that changes the numerator
return *this; // return the incremented value
}
fraction operator ++(int)
{
fraction temp=*this; // copy the value before increment
numerator+=denominator; // add one to the fraction, that changes the numerator
return temp;// return the old value of the object
}
```

Overloading insertion and extraction operators

Overloading insertion(<<) operator and extraction (>>) operator is a must when you want to input and output objects using stream class library in the similar fashion as built in data types. For a class fraction we should be able to do the following

```
fraction f1, f2(2,3);

cin >> f1;
cout << f2;
```

Since the first operand is iostream object, the operator function will be friend function and

the declarations will be

```
friend ostream& operator<< (ostream &out, fraction &fract)
{
out << fract.numerator <<"/" << fract.denominator ;
return out;
}
friend istream& operator >> (istream &in, fraction &fract)
{
in >> fract.numerator ;
in >> fract.denominator;
return in;
}
```

Overloading the Assignment operator =

Assignment operator does member wise copying and built in assignment operator function works well except with the classes having pointer data members. In such cases one must explicitly overload assignment operator.

The vector class has dynamic data member hence the assignment operator need to be explicitly overloaded

```
const vector & operator = (const vector & rightvector)
{ if (this != rightvector)
{ delete [] vectorarray;
maxsize= rightvector.maxsize;
size = rightvector.size;
vectorarray= new int [maxsize];
for(int i=0; i< size; i++)
vectorarray[i]=rightvector.vectorarray[i];
}
return *this;
```

```
}
```
Overloading the Subscript operator []
The function to overload the operator [] for a class must be the member of the class. Furthermore, because an array can be declared as constant or nonconstant we need to overload the operator [] to handle both cases.

```
int & operator[](int index)
{
assert( 0<= index && index <size);
return vectorarray[index];
}
const int & operator[](int index) const
{
assert( 0<= index && index <size);
return vectorarray[index];
}
```

Set A

1. Define a class named Complex for representing complex numbers. A complex number has the general form a + ib, where a- the real part, b - the imaginary part are both real numbers and i2=-1. Define parameterized and default constructor. Overload +, - and * operators with usual meaning.

2. Define a class named Clock with three integer data members for hours, minutes and seconds. Define parameterized and default constructors. Overload increment and decrement operators appropriately. Overload extraction and insertion operators.

Set B

1. Define a class Message with two data members one character pointer and an integer storing length. Overload operator binary + to represent concatenation of messages, [] to return a character at a specific position and = to copy one Message object to another.

2. A Matrix has rows and columns which decide the number of elements in the matrix. We will implement a matrix class that can handle integer matrices of different dimensions. We can overload addition, subtraction and multiplication operator to carry out usual matrix addition, subtraction and multiplication.

Set C

1. Why return type in case of assignment operator function is a constant reference?
   Is it possible to overload new and delete operators?

2. Why are << and >> operators overloaded as friend functions?

3. Which operator should be overloaded to convert the Message to uppercase?

Assignment Evaluation

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

Practical In-charge

Assignment 4: Inheritance, Function overriding

Inheritance

Definition: Inheritance is one of the feature of Object Oriented Programming System (OOPs) it allows the child class to acquire the properties (the data members) and functionality (the member functions) of parent class.

What is child class?

A class that inherits another class is known as child class, it is also known as derived class or subclass.

What is parent class?

The class that is being inherited by other class is known as parent class, super class or base class.Implementing inheritance in C++:

Syntax for creating a sub-class which is inherited from the base class:

Syntax

```
class subclass_name : access_mode base_class_name
{
        //body of subclass
};
```

Here, subclass_name is the name of the sub class, access_mode is the mode in which you want to inherit thissub class for example: public, private, protected. And base_class_name is the name of the base class from which you want to inherit the sub class.

Modes of Inheritance

1. **Public mode:** If a sub class is derived from base class in public mode then the public member of the base class will become public in the derived class and protected members of the base class will become protectedin derived class. Private members of the base class will never get inherited in subclass.
2. **Protected mode:** If a sub class is derived from a base class in protected mode then both public member andprotected members of the base class will become protected in derived class. Private members of the base class will never get inherited in sub class.
3. **Private mode:** If a sub class is derived from a base class in private mode then both public member and

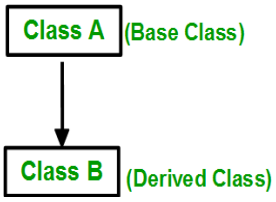
protected members of the base class will become Private in derived class. Private members of the base classwill never get inherited in sub class.

The following table summarizes the above three modes and shows the access specifier of the members ofbase class in the sub class when derived in public, protected and private modes:

	Derived Class	Derived Class	Derived Class
Base class	**Public Mode**	**Private Mode**	**Protected Mode**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

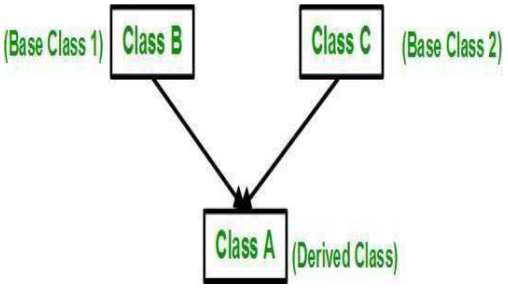
Types of Inheritance in C++

1. **Single Inheritance** : In single inheritance, a class is allowed to inherit from only one base class. i.e. one subclass is inherited by one base class only.



.

2. **Multiple Inheritance:** Multiple Inheritance is a feature of C++ where a class can inherit from more than oneclasses. i.e one **sub class** is inherited from more than one **base classes**.

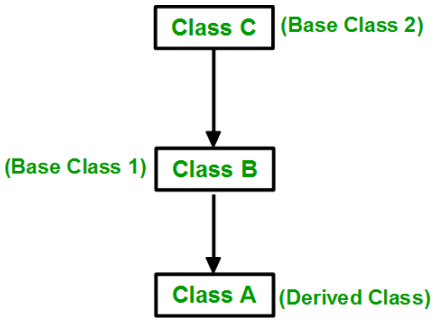


Syntax:

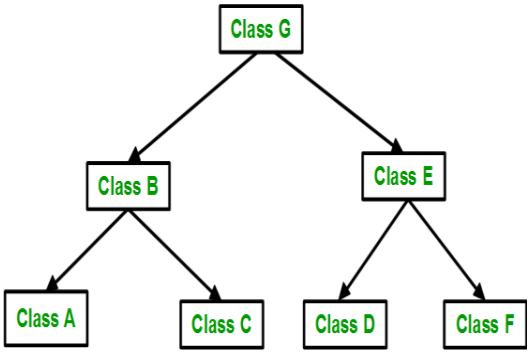
Class subclass_name : access_mode base_class1, access_mode base_class2, ...
{
//body of subclass
};

Here, the number of base classes will be separated by a comma („, „) and access mode for every base classmust be specified.

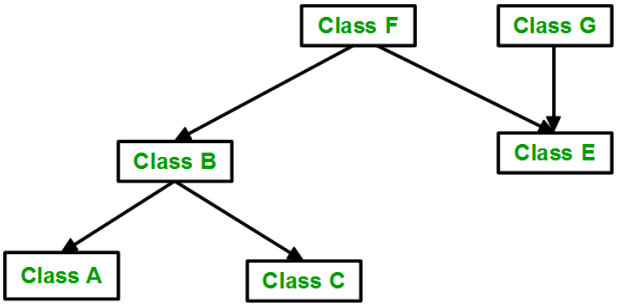
3. **Multilevel Inheritance** : In this type of inheritance, a derived class is created from another derived class.



4. **Hierarchical Inheritance**: In this type of inheritance, more than one sub class is inherited from a singlebase class. i.e. more than one derived class is created from a single base class.



5. **Hybrid (Virtual) Inheritance**: Hybrid Inheritance is implemented by combining more than one type ofinheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance. Below image shows the combination of hierarchical and multiple inheritance:



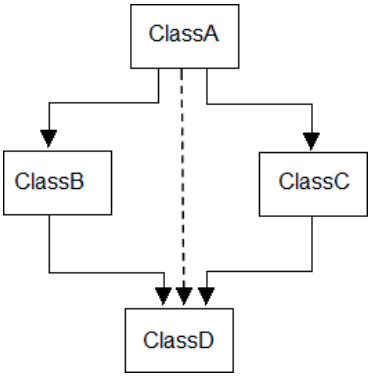
Virtual base class

An ambiguity can arise when several paths exist to a class from the same base class. This means that a childclass could have duplicate sets of members inherited from a single base class.

C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is takenso that the duplication is avoided regardless of the number of paths that exist to the child class.

Uses of virtual base class:

When two or more objects are derived from a common base class, we can prevent multiple copies of the baseclass being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class‟ name with the word virtual.



Syntax:

```
class A
{
        //body of class A

 }
        //body of class B
}
} class C : public virtual A
{
          //body of class C
}
class D : public B, public C
{
          //body of class D
}
```

Constructor in derived class

1) 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.
2) 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.
3) While applying inheritance, we usually create objects using derived class. Thus, it makes sense for the derivedclass to pass arguments to the base class constructor.
4) When both the derived and base class contains constructors, the base constructor is executed first and then theconstructor in the derived class is executed.
5) 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 theorder of inheritance.
6) The derived class takes the responsibility of supplying the initial values to its base class. The constructor ofthe 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.
7) A base class constructor is called and executed before executing the statements in the body of the derived class.

Syntax

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

Destructor in derived class

Destructors are invoked in the reverse order of the constructor invocation i.e. The destructor of the derivedclass is called first and the destructor of the base is called next.

Syntax

```
~Derived-Destructor()
{
// Body of Derived Destructor
}
```

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. Calculate total price, total discount and display bill using appropriate manipulators.

2. Implement the following class hierarchy. Student: id, name, StudentExam (derived from Student): Marks of n subjects (n can be variable) StudentResult (derived from StudentExam): percentage, grade Define a parameterized constructor for each class and appropriate functions to accept and display details. Create n objects of the StudentResult class and display the marklist using suitable manipulators.

3. Create three classes Car, Maruti and Maruti800.Class Maruti extends Car and Class Maruti80 0 extendsMaruti.Maruti800 class is able to use the methods of both the classes (Car and Maruti).

4. Design a two base classes Employee (Name, Designation) and Project(Project_Id, title). Derive a classEmp_Proj(Duration) from Employee and Project. Write a menu driven program to Build a master table. Display a master table Display Project details in the ascending order of duration.

5. Create a Base class Flight containing protected data members as Flight_no, Flight_Name. Derive a classRoute (Source, Destination) from class Flight. Also derive a class Reservation(Number_Of_Seats, Class, Fare, Travel_Date) from Route. Write a C++ program to perform following necessary functions:
   Enter details of "n" reservations
   Display details of all reservations
   Display reservation details of a Business class

Set B

1. Create a base class Account (Acc_Holder_Name, Acc_Holder_Contact_No). Derive two classes as Saving_Account(S_Acc_No., Balance) and Current_Account(C_Acc_No., Balance) from Account. The savings account provides interest and withdrawal facility. The current account provides no interest facility. Saving account maintains a minimum balance of 2000 and if the balance falls below this level, aservice charge of Rs 500 is imposed. Current account maintains a minimum balance of 5000 and if thebalancefalls below this level, a service charge of Rs 1000 is imposed.
   Write a C++ menu driven program to perform following functions:
   Accepting Amount anddeposit it into account
   Withdrawing amount from account.
   Calculating Interest and service charge where interest rate for Saving account is 10% ofbalance Displaying information of account.

2. Create a Base class Train containing protected data members as Train_no, Train_Name. Derive a class Route(Route_id,Sorce,Destination) from Train class. Also derive a class Reservation (Number_of_Seats,Train_Class,Fare,Travel_Date) from Route.
   a to perform following necessary functions:
   a. Enter details of „n" reservations
   b. Display details of all reservations
   c. Display reservation details of a specified Train class

Set C

1. Create two base classes Learn_Info(Roll_No, Stud_Name, Class, Percentage) and Earn_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

Assignment Evaluation

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

Practical In-charge

Assignment: 05

Virtual functions, Pure virtual function

Virtual Functions:

What is Virtual Functions?

A virtual function is a member function within the base class that we redefine in a derived class. It isdeclared using the virtual keyword.
When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs.

Rules for Virtual Function in C++:

1) They are always defined in a base class and overridden in derived class but it is not mandatory to override inthe derived class.
2) The virtual functions must be declared in the public section of the class.
3) They cannot be static or friend function also cannot be the virtual function of another class.
4) The virtual functions should be accessed using a pointer to achieve run time polymorphism.

```
class Base
{
int a;
public:
        virtual void func();        //function in base class
};
class Dervied : public Base
{
int b;
public:
        void func();       //function in derived class
}
```

The main thing to note about the program is, derived class function is called using a base class pointer. The idea is, virtual functions are called according to the type of object pointed or referred, not according to the type of pointer or reference. In other words, virtual functions are resolved late, at runtime.

Example:

```
#include<iostream.h>
class Base
        {

      public:
        virtual void show() { cout<<" In Base \n"; }
        };

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

        int main(void)
        {
                Base *bp = new Derived;
                bp->show(); // RUN-TIME POLYMORPHISM
                return 0;

                }
```

Virtual functions in derived classes:

In C++, once a member function is declared as a virtual function in a base class, it becomes virtual in everyclass derived from that base class.

Example:

For example, the following program prints "C::fun() called" as B::fun() becomes virtual automatically.

```
#include<iostream>
 class A
{
  public:
    virtual void fun()
    { cout<<"\n A::fun() called ";}
};
 class B: public A {
  public:
    void fun()
    { cout<<"\n B::fun() called "; }
};

class C: public B {
  public:
    void fun()
    { cout<<"\n C::fun() called "; }
                };
```

```
int main()
{
  C c; // An object of class C
  B *b = &c; // A pointer of type B pointing to c
  b->fun(); // this line prints "C::fun() called"

  return 0;
}
```

Pure Virtual Function and Abstract Class:

What is pure virtual function?

A pure virtual function is a type of function, which has only a function declaration. It does not have thefunction definition.

Abstract class:

An abstract class is one that is not used to create object. An abstract class is designed to act as a base class (tobe inherited by other classes). The class, which contains pure virtual functions, is called as abstract class.

Syntax:

```
class Base
{
        inti;
        public:
                virtual void getData()=0;
};
class Derived :
public Base
{};
```

Example:

```
#include<iostream.h>
class BaseClass
{
        Public:
                virtual void Display1() = 0; // Pure virtual function
                virtual void Display2() = 0; // Pure virtual function

                void Display3()
                {
                        cout<< "\n\t This is Display3() method of Base Class;
                }
};
class DerivedClass : public BaseClass
{
        public:
                void Display1()
                {
                        cout<< "\n\t This is Display1() method of Derived class";
                }
                void Display2()
                {
                        cout<<"\n\t This is Display2() method of Derived Class";
                }
};
void main()
{
        DerivedClass D;
        D.Display1();           //This will invoke Display1() method of Derived Class
        D.Display2();           //This will invoke Display2() method of Derived Class

        D.Display3();           //This will invoke Display3() method of Base Class
}
```

Set A

1. Write a C++ program to create a class shape with functions to find area of the shapes and display the nameof the shape and other essential component of the class. Create derived classes circle, rectangle and trapezoid each having overridden functions area and display. Write a suitable program to illustrate virtualfunctions.

2. A book (ISBN) and CD (data capacity) are both types of media (id, title) objects. A person buys 10 mediaitems, each of which can be either book or CD. Display the list of all books and CD"s bought. Define theclasses and appropriate member functions to accept and display data. Use pointers and concepts of polymorphism (virtualfunctions).

3. Create a base class Student (Roll_No, Name) which derives two classes Theory and Practical. Theory class contains marks of five Subjects and Practical class contains marks of two practical subjects. Class Result (Total_Marks, Class) inherits both Theory and Practical classes. (Use concept of Virtual Base Class)
   Write a C++ menu driven program to perform the following functions: Build a master tableDisplay a master table
   Calculate Total_marks and class obtained

Set B

1. Write a program with Student as abstract class and create derive classes Engineering, Medicine and Science from base class Student. Create the objects of the derived classes and process them and access them using array of pointer of type base class Student.

2. Create a class called LIST with two pure virtual function store() and retrieve(). To store a value call storeand to retrieve call retrieves function. Derive two classes stack and queue from it and orverride store andretrieve.

3. Create a base class Person (P_Code, P_Name). Derive two classes Account(Ac_No., Balance) and Official(Designation, Experience) from Person. Further derive another class Employee from both Accountand Official classes. (Use Concept of Virtual BaseClass)
   Write a C++ menu driven program to perform the following functions: Build a master table for Display a master table of "n" employees. Display employees whose designation is H.O.D.

4. 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 BaseClass)
   Write a C++ menu driven program to perform the following functions: To Accept and display student details Calculate Subject wise total marks obtained. 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

Set C

1. Write a program to design a class representing the information regarding digital library (book, tape: book and tape should be separate classes having the base class as media). The class should have functionality foradding new items, issuing and deposit etc.

2. Write a program to calculate bonus of the employees. The class master derives the information from bothadmin and account classes which in turn derives information from class person. Create base and all derived classes having same member functions call getdata, display data and bonus. Create a pointer of base class that capable of accessing data of any class and calculates bonus of the specified employee. (Hint: Use virtual functions)

Assignment Evaluation

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

Practical In-charge