Object Oriented Programming using C++

[Lab Planner]

Academic Year 2011-12

Indira Gandhi Institute of Technology

Guru Gobind Singh Indraprastha University

Kashmere Gate, Delhi

Table of Contents

0. Guidelines

This document titled ‘Lab Planner’ is written based on following guidelines.

1. Lab work is divided into across weeks. Each week has certain objectives to be accomplished by solving programming problems of varying complexity.
2. Each week has 4 lectures supplemented with 2 hours scheduled Lab work.
3. We learn programming by *practice*. Being a course on programming, it is expected and strongly encouraged that **students not only utilize their Lab hours efficiently, but also configure programming environment in their personal computing environment**.
4. Lab exercises are comprehensive and are created in order to emphasis ***rigor*** in the programming course.

1. Syllabus

As taken from www.ipu.ac.in website*.

**Academics → Academics → Scheme Syllabus (Affiliated Institutes, w.e.f. 2005-06)*

UNIT – I

Introduction: Introducing Object-Oriented Approach related to other paradigms (functional, data decomposition), Characteristics of Object-Oriented Languages.

Basic terms and ideas: Abstraction, Encapsulation, Information hiding, Inheritance, Polymorphism, Review of C, Difference between C and C++, cin, cout, new, delete operators. **[No. of Hrs: 11]**

UNIT – II

Classes and Objects: Abstract data types, Object & classes, attributes, methods, C++ class declaration, State identity and behavior of an object, Constructors and destructors, instantiation of objects, Default parameter value, Copy Constructor, Static Class Data, Constant and Classes, C++ garbage collection, dynamic memory allocation. **[No. of Hrs. 11]**

UNIT – III

Inheritance and Polymorphism: Inheritance, Types of Inheritance, Class hierarchy, derivation – public, private & protected, Aggregation, composition vs classification hierarchies, Polymorphism, Type of Polymorphism – Compile time and runtime, Method polymorphism, Polymorphism by parameter, Operator overloading, Parametric polymorphism, Generic function – template function, function name overloading, Overriding inheritance methods **[No. of Hrs: 11]**

UNIT – IV

Files and Exception Handling: Persistant objects, Streams and files, Namespaces, Exception handling, Generic Classes

Standard Template Library: Standard Template Library, Overview of Standard Template Library, Containers, Algorithms, Iterators, Other STL Elements, The Container Classes, General Theory of Operation, Vectors. **[No. of Hrs: 11]**

TEXT BOOKS:

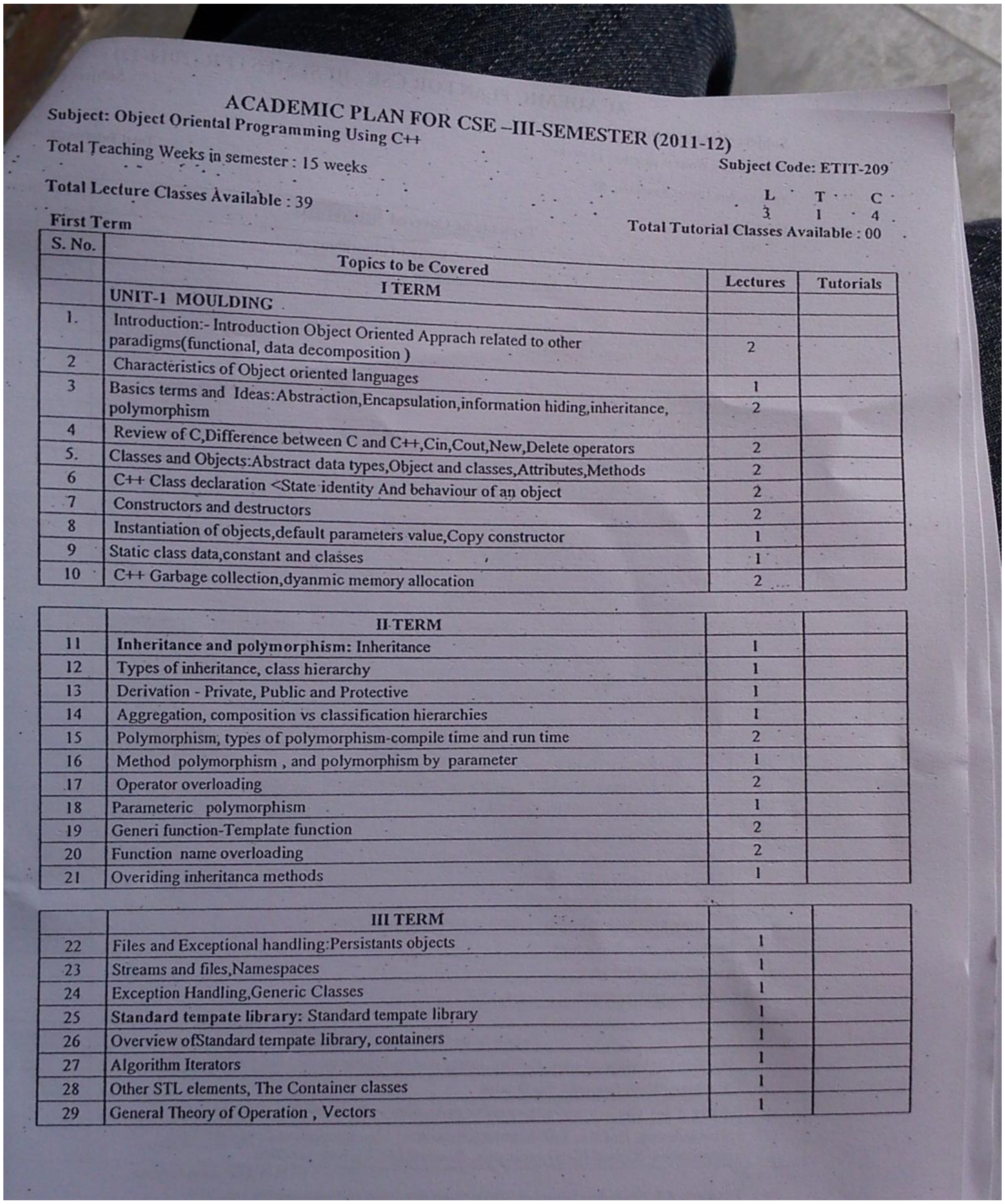
1. A.R.Venugopal, Rajkumar, T. Ravishanker "Mastering C++", TMH, 1997.
2. R. Lafore, "Object Oriented Programming using C++", BPB Publications, 2004.
3. Schildt Herbert, "C++: The Complete Reference", Wiley DreamTech, 2005.

REFERENCE BOOKS:

1. D . Parasons, "Object Oriented Programming with C++", BPB Publication, 1999.
2. Steven C. Lawlor, "The Art of Programming Computer Science with C++", Vikas Publication, 2002.
3. Yashwant Kanethkar, "Object Oriented Programming using C++", BPB, 2004.

1.1 Course Coverage Plan

This section contains the topics to be covered for various examinations (Minor-1, Minor-2 and End Semester Exam).



ACADEMIC PLAN FOR CSE –III-SEMESTER (2011-12)

Subject: Object Oriental Programming Using C++

Total Teaching Weeks in semester : 15 weeks

Subject Code: ETIT-209

Total Lecture Classes Available : 39

L	T	C
3	1	4

Total Tutorial Classes Available : 00

First Term

S. No.	Topics to be Covered	Lectures	Tutorials
	I TERM		
	UNIT-1 MOULDING		
1.	Introduction:- Introduction Object Oriented Apprach related to other paradigms(functional, data decomposition)	2	
2	Characteristics of Object oriented languages	1	
3	Basics terms and Ideas:Abstraction,Encapsulation,information hiding,inheritance, polymorphism	2	
4	Review of C,Difference between C and C++,Cin,Cout,New,Delete operators	2	
5.	Classes and Objects:Abstract data types,Object and classes,Attributes,Methods	2	
6	C++ Class declaration <State identity And behaviour of an object	2	
7	Constructors and destructors	2	
8	Instantiation of objects,default parameters value,Copy constructor	1	
9	Static class data,constant and classes	1	
10	C++ Garbage collection,dyanmic memory allocation	2	
	II TERM		
11	**Inheritance and polymorphism:** Inheritance	1	
12	Types of inheritance, class hierarchy	1	
13	Derivation - Private, Public and Protective	1	
14	Aggregation, composition vs classification hierarchies	1	
15	Polymorphism, types of polymorphism-compile time and run time	2	
16	Method polymorphism , and polymorphism by parameter	1	
17	Operator overloading	2	
18	Parameteric polymorphism	1	
19	Generi function-Template function	2	
20	Function name overloading	2	
21	Overiding inheritanca methods	1	
	III TERM		
22	Files and Exceptional handling:Persistants objects	1	
23	Streams and files,Namespaces	1	
24	Exception Handling,Generic Classes	1	
25	**Standard tempate library:** Standard tempate library	1	
26	Overview ofStandard tempate library, containers	1	
27	Algorithm Iterators	1	
28	Other STL elements, The Container classes	1	
29	General Theory of Operation , Vectors	1	

2. Lab Work Guidelines and Plan

Guiding Principle:

The only way to learn a programming language is to write programs in it solving various problems. The more we practice and get our hands dirty, the more we learn. And as we move on to solve increasingly challenging problems, we start *enjoying* programming.

3.1 Problem Categorization

Being a programming intensive course, the programming problems are categorized into the following:-

3.1.1 Basic problems

These are the set of problems that are *mandatory*. Each week, at least around **five** problems from this category MUST be completed within the stipulated Lab hours every week and their completion need to be reported to the Lab Instructor.

3.1.2 Advanced problems

These are the set of problems that are *of advanced nature* to be solved in order to become proficient in the programming language and to better appreciate the powerful techniques offered by the language. It is *mandatory* that a student solves at least around **two** problems from this category every week. The stipulated lab hours wouldn't be sufficient and it is expected that student should spend time at home to solve these problems.

3.1.3 Bonus problems

These are extra set of problems that will be given occasionally. Students are *highly encouraged* to solve them only after completing minimum requirements of above categories of problems. Solving bonus problems will fetch students extra bonus scores which shall help her in earning better grades.

3.2 Problem Evaluation Plan and Submission Guidelines

The underlying guideline for programming problem submission and evaluation is to minimize the overhead for the students (so that they concentrate on solving problems) and to maximize the continuous assessment.

Following are the modalities in view of the above guiding principle:-

1. All programs for solving a programming problem will be submitted **online to a web server** hosted in the Lab.

2. Programs submitted will undergo *online* compilation and *offline* (optional) testing along with necessary *plagiarism checks* in order to evaluate the program from time to time.

3. Students would maintain a handwritten *lab file* to keep a record of their *weekly progress*. For each programming problem, writing the problem statement and **only the most relevant code snippet** is required to be written in the *lab file.*

4. **PRINTOUTS OF PROGRAM CODE WILL NOT BE ACCEPTED. ONLY HAND WRITTEN LAB FILE SHOULD BE MAINTAINED.**

5. There would be **zero tolerance** for plagiarism of any kind. In any event of detection of plagiarized code, all students involved would be penalized including the perceived "original" writer. It is responsibility of a student to secure their code.

6. Grades would fall under following categories:

    a. Excellent: EX, students who have been regularly completing all the programming problems (including all advanced and bonus problems), and not just restricting themselves to minimum criteria as mentioned in Section 3.1.

    b. Very Good: A, students who have been regularly completing most of the programming problems (including all advanced and bonus problems), and not just restricting themselves to minimum criteria as mentioned in Section 3.1.

    c. Good: B, students who have been regularly completing all of the programming problems as per the minimum criteria and occasionally solving bonus problems.

    d. Average: C, students who have been almost regularly completing most of the programming problems as per the minimum criteria.

    e. Poor: D, students who have been irregularly completing most of the programming problems as per the minimum criteria as mentioned in Section 3.1.

7. Range of marks (in percentage) corresponding to above mentioned grades are *tentatively* as follows:

Grade	Range of Marks	Remarks
EX	95-100	Excellent
A	85-94	Very Good
B	75-84	Good
C	65-74	Average
D	55-64	Poor

3. Lab Exercises

This section includes a consolidated list of programming exercises based on topics covered every week. Division of exercises among Lab Problems, Submission Based Problems and Bonus Problems shall be announced from time to time.

Note: Please ensure that you have installed the programming environment as mentioned in Appendix A.

3.1 Week-1: Familiarity with Programming Environment

Basic Problems:

1. Write a program to print 'Hello World'.

2. Write a program to read an employee's information from the user and print the same. Employee's information shall include employee ID (*int*), employee name (*string*) and employee salary (*float*).

3. Write a program to take two integer inputs and output their sum, difference, product and division (quotient and remainder) as result based on a third input (operator).

4. Write program(s) to perform following conversions (and vice-versa):-
    a. Temperature in Celsius to Fahrenheit
    b. Height in Centimeters to Feet and Inches

Your program(s) should take care that the output is *formatted* in any *format* chosen by you.

5. A *perfect number* is one whose divisors add up to the number. For example, 6 is a perfect number because 6 = 1 + 2 + 3. Write a program that prints all perfect numbers from 1 till 10000.

Advanced Problems:

1. Write a program to print all *possible rearrangements* of a given string input. For instance, if input string is "abc", output is "abc", "acb", "bac", "bca", "cab" and "cba".

2. Amicable Numbers: Two numbers N1 and N2 are called *amicable* if sum of proper divisors of N1 is equal to that of N2 and vice-versa. For instance, (220,284) are a pair of *amicable* numbers because sum of proper divisors of 220 (1+2+4+5+10+11+20+22+44+55+110=284) is 284 and sum of proper divisors of 284 (1+2+4+71+142=280) is 280. Write a program to output all *amicable* number pairs from 0 till 100000.

Bonus Problems:

1. Omit out certain text from the "Hello World" program done above and find out at least FIVE error messages. Also mention the type of error (compile or link or load) occurred.

2. How is the "Hello World" program different than a corresponding C program, give both, syntactically and semantic differences? Enlist your differences.

3.2 Week-2: Data Types

Basic Problems

1. Write a program that returns a Boolean data type (True, False) based on whether a given number is prime or not. Use this to find all prime factors of a given input number. Express the input number as a product of its prime factors. For instance for an input number 24, express output as 24 = 2 x 2 x 2 x 3.

2. Every integer number is represented in computer in binary form as a sequence of 0's and 1's. Write a program that takes an input integer and outputs number of 0's and 1's, respectively in its binary representation form. For example, binary equivalent of decimal number 12 is 1100, so it has 2 0's and 2 1's.

3. Write a program which performs the following tasks:-

a. Encoding: Takes a character array as input and encodes it to an output array based on the following conversion formula:-

out(i) = in(i) + X,

where 'i' is the ith character in the array and 'X' is a random number between 1 and 26

b. Decoding: Takes encoded array of character and X as inputs and outputs the decoded (original) character array.

4. A hexadecimal number is represented with combinations of 0 to 9, 'a' through 'f', and/or 'A' through 'F'. Write a program that takes input string of hexadecimal digits and gives the output as its equivalent integer value. The allowable input digits are combinations of 0 to 9, 'a' through 'f', and 'A' through 'F'. Extend the program to accept input with optional prefix of '0x' or '0X'. Further extend the program to accept input in octal form and output its decimal equivalent.

5. Write a program that finds the size limits of all data types available in C++. Use <*limits*> header file to write the program.

6. Write a program that finds number of days between two input dates give in dd-mm-yyyy. Use enumeration to store all months of year.

3.3 Week-3: Operators, Statements, Pointers and Arrays

Basic Problems

1. Write a program to find values of each of the following expression (execute each expression in the same order as given below), assume following initial values of the variables:

a = 1, b = 2, c =3 , d =4, x = 1, i = 5.

*a = b + c * d << 2 & 8*

a & 0 7 7 != 3
a == b || a == c && c < 5
c = x != 0
0 <= i < 7
a = b == c ++
a = b = c = 0

2. Write a program that takes as input two bit strings and a bitwise operator as input and outputs the resulting bit string based on bitwise operator as input. Include support for all bitwise operators in your program.

3. Telephone directory is to be maintained with each individual's details recorded as {name, telephone number}. Write a program that accepts any number of such records enters by the users and then displays them in alphabetical order with respect to name of the individual. Make use of dynamic memory allocation using *new* and *delete* operators.

4. Keep reading a sequence of words as input from the user until user enters a word *quit.* Output the words entered removing the duplicate words and arranging them in increasing order of their length, alphabetically sort words of same length.

5. What are the differences between pointer and reference. Illustrate using code snippets.

3.4 Week-4: Functions and Structures

Basic Problems

1. Write a program which takes as input a *string* and converts it into its equivalent *number* and returns the same.

2. Create a *structure point* with X-coordinate and Y-coordinate and perform the following geometric operations using *functions*:-
    a. Given two end points of line segment, find length of line segment.
    b. Given a circle C (centre point and radius) and a point P, find if point P lies inside the circle or outside.
    c. Given two line segments, find if they are parallel or not.
    d. Given all three coordinates of a triangle, verify if a valid triangle can be formed, if yes then calculate area of triangle.

3. Write a program that calculates area and perimeter of the following geometric figures. Your program should use *function overloading* and each function should take as inputs the required arguments (without constraining the user) and return both area and perimeter (make use of an appropriate *structure* to do so).
    a. Square
    b. Circle
    c. Rectangle

d. Triangle

4. Write a function *myAvg( )* that takes *unspecified* number of input arguments and finds the average of all input numbers and returns the same. Your function should have at least one argument which shall decide the number of input numbers for whom average is to be calculated. Once defined the function can be called as below,

    avg = myAvg (3, 10, 20, 30); *// avg = (10+20+30)/3*
    avg = myAvg (5, 10, 20, 30, 40, 50); *// avg = (10+20+30+40+50)/5*

5. Write a program *myProg.cpp* that takes input strings (student names) as *command line arguments* and outputs a greeting, message to all of them. The program once compiled into *myProg* should run as below at command line:

    ./myProg neha kirti divya
    Hello Neha, Kirti and Divya, Welcome to IGIT! *// Output*

7. Write a program with *recursive functions* that perform the following:-
    a. Reverse an input string
    b. Check if an input string is palindrome or not
    c. Binary search
    d. Sorting by putting least number at the beginning in each successive calls

8. What is the scenario where pointers to functions can be put to use. Illustrate with an example program.

9. How can we realize the use of default arguments through overloaded functions? Illustrate with an example program.

3.5 Week-5: Classes and Objects

Basic Problems

1. Write a program which provides concrete representation for the concept of date, using only *structures* and member functions. Provide functionality to initialize date, add day, add month and add year to the date *structure.*

2. Write a program which provides concrete representation for the concept of date, using only *classes* and member functions. Provide functionality to initialize date, add day, add month and add year to the date *class*.

3. Extend the date *class* to include different ways to initialize date namely initializing them with zero for day, month and year using *default constructor*, initializing them with given input values from the user using *parameterized constructor* and initializing using a user supplied input string with date formatted as “dd-mm-yyyy”.

4. Include the concept of a *default* date in date *class* using *static data members* and write a *static member function* which operates upon it.

5. Write a program which provides concrete representation for the concept of dynamic string *class* which provides support for dynamically allocated string. Write constructor(s) to allocate memory and destructor to deallocate memory for the string.

6. Student database is to be maintained in a college. Design suitable *class* and write member functions which provide the following functions:
   a. Insert details of student
   b. Search an existing student w.r.t. Registration number
   c. Modify the details of an existing student
   d. Delete a student
   e. Sort list of students w.r.t. Registration number and Name

For simplicity, assume that a student has following attributes only, namely, registration number (*long*) and name (*string*). Details of students is maintained in a linked list.

7. Extend the student database program to save all student information in a file. When the program is started for the first time, this file is *read* and information retrieved is filled in the linked list. All operations (insert, delete, modify) are performed on the linked list. Before exiting from the program, the current information in the linked list is saved in the file.

8. Various real world software involves handling of lists of related data, for example, list of students, list of teachers, etc. Write a program that captures the concept of list in form of a *class*, so that whenever in a real world software a list is required, an object of our *list class* could be used. Members functions of *list class* should perform all operations required by a linked list namely searching, inserting, deleting and updating.

9. What is the difference between the following object initializations?
   myDate d1(2,9,2011);
   myDate d1 = myDate(2,9,2011);
   *myDate *d1 = new myDate(2,9,2011);*

10. Differentiate between *physical* and *logical constness*? What is the use of *const* keyword in the context of member functions of the class? How does *mutable* keyword play its role in it? Illustrate with a suitable program.

11. What are the different ways in which objects of a class can be created? Explain with an illustrative program for each.

3.6 Week-6: Operator Overloading

Basic Problems

1. Write a program which provides concrete representation for the concept of complex number.

Using *operator overloading*, perform the operations of addition and multiplication of two given complex numbers. Implement operator overloading through both methods namely friend function and member function.

2. Extend the concept of complex number by providing provisions for unary operators + and – which increments and decrements both real and imaginary part of complex numbers by 1, respectively. Implement operator overloading through both methods namely friend function and member function.

3. Write a program which provides support for creation of *dynamic string*. Hereafter, implement the operations of following conventional C-string functions using the operator symbols as mentioned in table given below.

C-string Function	**Operator Symbol(s)**
strcpy (s1, s2)	s1 = s2
strcmp (s1, s2)	s1 == s2, s1 <= s2, s1 >= s2
strcat (s1, s2)	s1 + s2, *result stored in s1*

4. Write a program that provides support for the following conversions:-
   a. Height in cms *(int)* into feet and inches (*class with feet and inches)*
   b. Height in feet and inches (*class with feet and inches)* into cms (*int)*
   c. Height in cms (*class with cms)* into feet and inches (*class with feet and inches)*

5. Write a program which provides support for both prefix and postfix forms for ++ and – *operators* for a 2D point class.

6. Write a program that captures a 3D point. Your program should overload << *operator* and << *operator* to output and input the value of 3D point object, respectively.

3.7 Week-7: Inheritance

Basic Problems

1. Write a C++ program that represents an Vehicle-Car and Vehicle-Motorcycle relationship using **structures with nested members**. C++ program should provide following operations.
   a.Add a vehicle (car/motorcycle).
   b.Maintain a common list of vehicles (use static allocation) for both cars and motorcycles.
   c.Display the vehicles (including both cars and motorcycles) with respect to their registration ID (alpha-numeric).

Assume vehicle attributes as Vehicle Registration ID and Owner Name, while attributes of the cars and motorcycle.

2. Write a C++ program that represents an HOD-Faculty relationship in an academic institution

using **public inheritance**. C++ program should provide following operations, only through member functions, no constructor/destructor should be used.

a. Add a faculty or HOD.
b. Maintain a common list (use static allocation) for both HODs and Faculties.
c. Display all the Faculties (including HOD who is also a Faculty) with respect to their names.

Assume faculty attributes as Faculty ID and Faculty Name, while attributes of the HOD as Department Name.

3. Design a <u>single C++ program</u> illustrating the following concept of inheritance:-
   i. Multiple Inheritance
   ii. Multilevel Inheritance
   iii. Hierarchical Inheritance

4. Design a <u>single C++ program</u> illustrating the following concept of inheritance:-
   i. Public derivation
   ii. Private derivation
   iii. Protected derivation

5. **Organization:** Write a program which provides concrete representation for a typical organization hierarchy as given below.

Managing Director		
Vice President, Engineering & Technology	Vice President, Sales & Marketing	Vice President, Human Resource
Managers	Managers	Managers
Employees	Employees	Employees

Below is a description of the organization:-

a. Managing Director, Vice President and Managers are employees of the organization. Hereafter, the word 'employee' would mean any normal employee including the above special employees.
b. Each employee has a name, employee ID, email ID, Date of Birth and Date of Joining.
c. Each manager is heading a team with Team Name and a list of Team Members who are employees.
d. Each vice president heads a division with Division Name and list of Managers under the division.

e. Managing Director heads the organization with Organization Name and Organization's Registration Number (a unique alpha-numeric identifier).

Write a C++ program to capture the above requirements. Provide a facility for the following operations:-

i. Add an employee (of any type namely normal employee, manager, vice president, managing director) along with its details to the organization.
   Note: While adding, take care of the necessary "associations" of the employee with other employees as per the description given above, say, while adding a manager, enter the employees to be managed and VP that shall manage the Manager.
ii. Delete an employee from the organization along with its all "associations".
iii. Search an employee on the basis of name and employee ID.
iv. Modify details of an employee.
v. Display the list of employees, sort with respect to their Date of Joining, Date of Birth (calculate their age and display it as well), Name and Organizational Hierarchy.

3.8 Week-8: More on Inheritance

Basic Problems

1. Write a C++ program that illustrates the concept of **virtual base class**. Use the family tree relationship consisting of grandparent, parents and child to illustrate the same.

2. What is the order of calling of constructors and destructor in case of multilevel inheritance between some classes with names A, B and C. Write a C++ program to support your answer.

3. Write a C++ program that illustrates the use of **pure virtual function**. Create an **abstract class** *shape* which has *area ( )* and *perimeter ( )* as its member functions. These functions are overridden by inherited classes namely *rectange, square, triangle* and *circle.*

4. Is it mandatory for a derived class to have a constructor if base class has a constructor. Write a C++ program to support your answer.

5. Can we perform run time polymorphism on operator functions. Write a C++ program that illustrates the use of **virtual function** for operator functions.

3.9 Week-9: Special Classes in C++

Basic Problems

1. Write a C++ program that represents a Teacher-Student relationship using **composition classes**. Teacher-Student relationship can be understood as "every teacher ***has*** number of students". Assume following attributes of Teacher and Student:-

   Teacher

a. Teacher ID (numeric)
b. Name
c. Subject

Student

a. Student Registration ID (numeric)
b. Name
c. Subject
d. Marks

Write a *display ( )* function that prints the average marks secured by students under a given teacher.

2. Write a C++ program that maintains a dictionary using **aggregation classes**. C++ program should provide support for following classes.

a. Create an *alphabet* class which represents an alphabet of English language.
b. Create a *word* class that represents a word along with its meaning.
c. Alphabet class maintains a pointer to the lists of all words (instances of word class) entered by user.

Write a *main( )* function that keeps prompting a user to enter words and meanings, until prompted to stop. After all words have been entered, display them in alphabetically sorted format as is done in a dictionary.

3. Write a C++ program illustrating **container classes** that maintains a dynamic list of integers, say *myDynamicArray*. The class, thus created, should provide following facility:-

a. Insert an integer into the list.
b. Delete an integer from the list based on index and value of integer.
c. Use an overloaded [] operator to index into any integer.
d. Search for an integer.
e. Sort integers in the list.
f. Display the list.

3.10 Week-10: Templates

Basic Problems

1. Write a C++ program that represents a collection of elements of pre-defined or user defined types using **template class**. The template class should provide following features:-

a. Adding a new element.
b. Deleting a existing element.
c. Displaying the collection of elements.

2. Using the **template class** consisting of collection of elements with operations of adding, deleting and displaying elements, create a collection of students, each student with attributes as name and roll number.

3. Write a C++ program to illustrate the concept of **template function** by writing template function which performs the following tasks:-
    a. Addition of two given type
    b. Subtraction of one type with another type

Test the working of template functions with pre-defined types like int, float, etc.

4. Using the **template functions** for the operation of addition and subtraction, perform the string concatenation and string truncation operation, respectively for addition and subtraction in template function.

5. Can we have a **template function** as a member function of **template class.** Write a C++ program to support using answer.

3.11 Week-11: Input and Output

Basic Problems

1. User enters multiple lines on console, each line is separated by another by a newline (user presses an 'enter' keystroke to differentiate between two lines). Write a C++ program that reads such an input from console outputs the line of maximum length.

2. User enters a paragraph across multiple lines on console. Write a C++ program that reads such an input from console outputs the number of characters, words and lines in the entered paragraph.
    where,
        'character' includes letters from English language, numerals and special symbols.
        'word' refer to sequences of 'characters' (defined above) separated by space or tabs or full stop.
        'line' refer to the words between two full stops.

3.12 Week-12: Exception Handling

Basic Problems

1. Division by zero is an exception. Write a C++ program to detect such an exception and handle it by displaying an appropriate message and exit from the program.

2. Square root of negative number is to be treated as an exception condition. Write a C++ program that handles such an exception by repeatedly prompted a user to keep entering until a positive number is fed as input and square root calculated subsequently.

3. Write a C++ program to illustrate the concept of **re-throwing an exception.**

3. In case of multiple nested functions, an exception occurring at a deeply nested function

causing **unwinding of stack**, write a C++ program to illustrate the same.
.
4. Exceptions are perhaps the only mechanism to handle errors inside constructors and destructors. Write a C++ program handling exception inside a constructor and destructor.

3.13 Week-13: Standard Template Library

Basic Problems

1. **Merge sort.** Write a program in C++ using *vector* class that inputs two sets of *random* integers (size of each input set to be provided by user), stores (inserts) them in two *vectors* ensuring a *sorted order* (use STL's *sort* algorithm to perform sorting) and finally merges the two sorted sets into a single set (in a third *vector* class object).

2. **Priority queue.** Design a C++ class that captures the concept of a process in an operating system. Each process may have attributes as unique process ID, process name and process priority number (lower value means higher priority). Using *dequeue* class, write C++ program to perform the operations of inserting a new process (with random attributes) and deleting a process with highest priority.

3. **Student list.** Design a C++ class that captures the concept of a student with attributes student ID, student name and student marks. Use *list* class to maintain a collection of students and provide operations for adding, deleting and updating attributes of a student. Use STL's pre-defined algorithm functions to search for a particular student, arrange student list in sorted order w.r.t name/ID and to find student scoring maximum/minimum marks.

4. **Set operations**. Using *set* class, illustrate all set operations namely union, intersection, subset, disjoint and membership. Use collection of *random* integers as input in the sets.

5. **Telephone directory.** A telephone directory consisting of unique telephone numbers along with the name of the telephone holder is to be maintained. People with similar names with different telephone numbers is possible but not the other way around. Write a C++ program using *map* class to maintain such a telephone directory and provide operations to add, delete, update and query (given a telephone number, report the telephone holder's name and vice versa) in such a directory. Store telephone records such that it facilitates such query operations, use STL's pre-defined algorithms to perform any compuation.

6. **Dictionary.** A dictionary consisting of words with their meanings is to be maintained. Same word can have more than one meaning. Write a C++ program using *mmap* class to maintain such a dictionary and provide operations to add, delete, update and query (given a word find its meaning) in such a dictionary. Store dictionary records such that it facilitates such query operations, use STL's pre-defined algorithms to perform any compuation.

Appendix A: Setting up programming environment

Aim:

Setting up Linux environment in Windows Operating System using Virtual Box Software

Step 1: Installation of Virtual Box

1. Download Virtual Box Software
http://download.virtualbox.org/virtualbox/4.1.2/VirtualBox-4.1.2-73507-Win.exe

2. Run the .exe file downloaded in Step 1 and install Virtual Box Software using the default options appearing during the installation procedure.

Note: By default, Virtual Box will install in C: drive, recommended free space that should be available in C: drive is around 15GB, if you don't have this much free space, select a different drive during the Virtual Box installation.

Step 2: Creating a Virtual Machine and installing Guest Operating System (ubuntu-10.04)

1. Download guest operating system (ubuntu-10.04 desktop i386)
http://www.psychocats.net/ubuntu/getting

2. Follow the steps in link below for creating virtual machine and installing guest operating system
http://www.psychocats.net/ubuntu/virtualbox

Step 3: Installing Virtual Box Guest Additions

1. Download virtual box guest additions
http://download.virtualbox.org/virtualbox/4.1.2/VBoxGuestAdditions_4.1.2.iso

2. Attach the guest addition iso file to CD/DVD device as explained in Step 2's 2. Next follow steps are given below
http://helpdeskgeek.com/linux-tips/install-virtualbox-guest-additions-in-ubuntu/

Step 4: Installing Software Packages (compiler, editor, etc.)

1. Go to Applications → Accessories → Terminal and type following at command prompt to install packages build-essential and vim editor.

```
sudo apt-get install build-essential
sudo apt-get install vim
```

Reference Online and Books

- C++ Programming Language by E. Balaguruswamy
- C++ Programming Language (Addison-Wesley) by Bjarne Stroustrup (Creator of C++)
- C++ Reference Library: http://www.cplusplus.com/reference
- C++ Tutorial: http://www.learncpp.com