WINTER – 2018 EXAMINATION
MODEL ANSWER

Subject: Object Oriented Programming **Subject Code:** 17432

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.
8) Program or code snippet shall be considered as an Example.

Q. No	Sub Q.N.	Answer	Marking Scheme
1.	**A)**	**Attempt any six of the following:**	**12**
	a)	**Write structure of C++ program.**	**2M**
	Ans.	**Structure of a C++ program** INCLUDE HEADER FILES CLASS DECLARATION MEMBER FUNCTION DEFINITIONS MAIN FUNCTION PROGRAM **OR** **Description:-** **1. Include header files:** Programmer include all header files which are require to execute given program such as *iostream.h* **2. Class declaration:** Programmer declares all classes which are necessary for given program. **3. Member Function definitions:** Programmer declares and defines member functions of a class. **4. Main Functions:** Programmer creates object and call various functions declared within various classes.	***correct structure 2M***

	b) **Ans.**	**Define pointer variable. Give its syntax.** **Definition:** Pointer is a variable that holds memory address of another variable of similar data type. **Syntax to declare pointer variable*:*** data_type *pointer_variable;	**2M** ***Correct definition 1M*** ***Correct syntax 1M***

	c)
	Ans.
		State any two access specifier with example.
		Access specifier:
		1. private:
		2. protected:
		3. public
		
		Example:
		class sample
		{
		private:
		int a;
		protected:
		int b;
		public:
		void display()
		{
		cout<<a<<b;
		}
		};
			2M
			
			List two correct access specifier 1M
			
			Any example 1M

	d) **Ans.**	**Define constructor. State any two types of constructor.** **Definition:** A constructor is a special member function whose task is to initialize the objects of its class. **Types of constructor:** 1) Default constructor 2) Parameterized constructor 3) Copy Constructor 4) Constructor with default value 5) Multiple constructor/overloaded constructor	**2M** ***Correct definition 1M*** ***List Any two types 1M***

WINTER – 2018 EXAMINATION
MODEL ANSWER

Subject: Object Oriented Programming **Subject Code:** 17432

	e) **Ans.**	**Define polymorphism. List types of polymorphism.** **Definition:** Polymorphism means ability to take more than one form that means a program can have more than one function with same name but different behavior. **Types of polymorphism:** 1) Compile time polymorphism 2) Runtime polymorphism	**2M** ***Correct definition 1M*** ***List two types 1M***
	f) **Ans.**	**Write any two advantages of inheritance.** **Advantages of inheritance:** 1) Use of inheritance in a program gives reusability of code. 2) Inheritance avoids duplication of code in program. 3) Inheritance reduces length of code. 4) Inheritance reduces time to compile the lengthy code by reusing it.	**2M** ***Any two relevant advantages 1M each***
	g) **Ans.**	**Explain the concept of this pointer.** **Concept of this pointer:** C++ use a unique keyword called "this" to represent an object that invokes a member function. This unique pointer is automatically passed to a member function when it is invoked. "this" is a pointer that always points to the object for which the member function is called.	**2M** ***Correct explanation 2M***

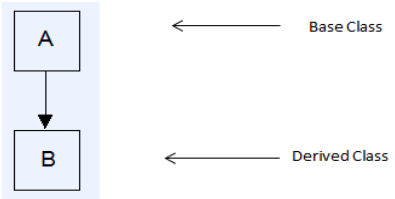
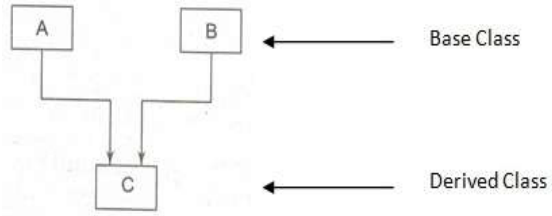
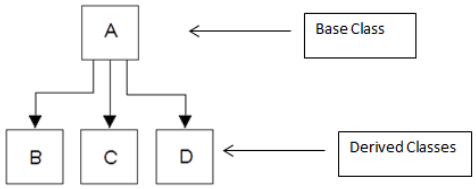
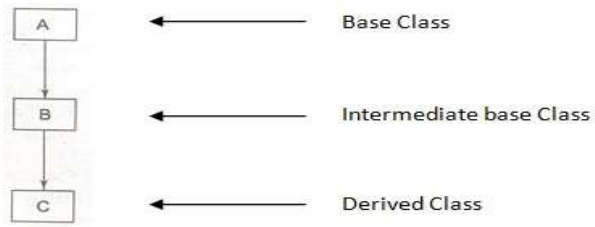
	h) **Ans.**	**Modify the given code to make its constructor with default argument**
		class add
		{
		private :
		int a:
		Public : add (int x)
		{
		a = x:
		}
		};
		class add {
		private:
		int a;
		public:
		add (int x=**1**)
		{
		a=x;
		}
		};
			2M
			
			Correct code 2M
			
			Any default value shall be considered

WINTER – 2018 EXAMINATION
MODEL ANSWER

Subject: Object Oriented Programming **Subject Code:** 17432

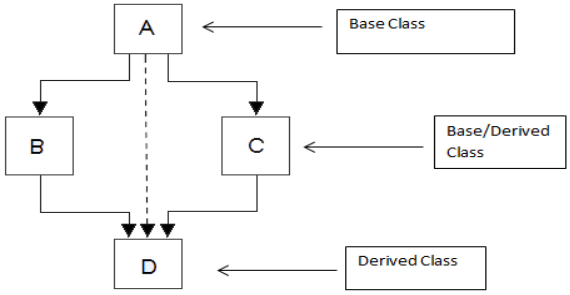
1.	**B)**	**Attempt any two of the following:**	**8**
	a)	**Explain the concept of overloaded constructor in a class with suitable example.**	**4M**
	Ans.	**Overloaded constructor:** When more than one constructor function is defined in a same class then it is called as overloaded constructor. All constructors are defined with the same name as the class name they belong to. Each of the constructors contains different number of arguments or different data type of arguments. Depending upon the number of arguments and their data type, the compiler executes appropriate constructor.	***Correct Explanation 2M***

		Example:-
		#include<iostream.h>
		#include<conio.h>
		class integer
		{
		int m, n;
		public:
		integer()
		{
		m = 0;
		n = 0;
		}// constructor 1
		integer(int a, int b) {
		m = a;
		n = b;
		cout<<"value of m="<<a;
		cout<<"value of n="<<b;
		} // constructor 2
		}
		void main() {
		clrscr();
		integer i1;
		integer i2(20,40);
		getch();
		}
		In the above example, constructor is overloaded by defining two constructors in the same class. Both the definitions are different with respect to number of arguments. The first constructor does not accept any argument and the second constructor accepts two integer arguments.
			Correct Example 2M

	b)	**List different types of inheritance with suitable diagram.**	**4M**
	Ans.	**Types of inheritance:** 1) **Single inheritance**: A derived class is derived from only one base class. **Diagram:**  2) **Multiple inheritance**: A derived class is derived from more than one base classes. **Diagram**:  3) **Hierarchical inheritance**: More than one derived classes are derived from single class. **Diagram:**  4) **Multilevel inheritance**: A derived class is derived from a derived class (intermediate base class) which in turn derived from a single base class. **Diagram:**  	***Any four types of inheritance with suitable diagram 1M each***

5) **Hybrid inheritance**: Combination of single, multiple, multilevel and hierarchical inheritance.

Diagram:



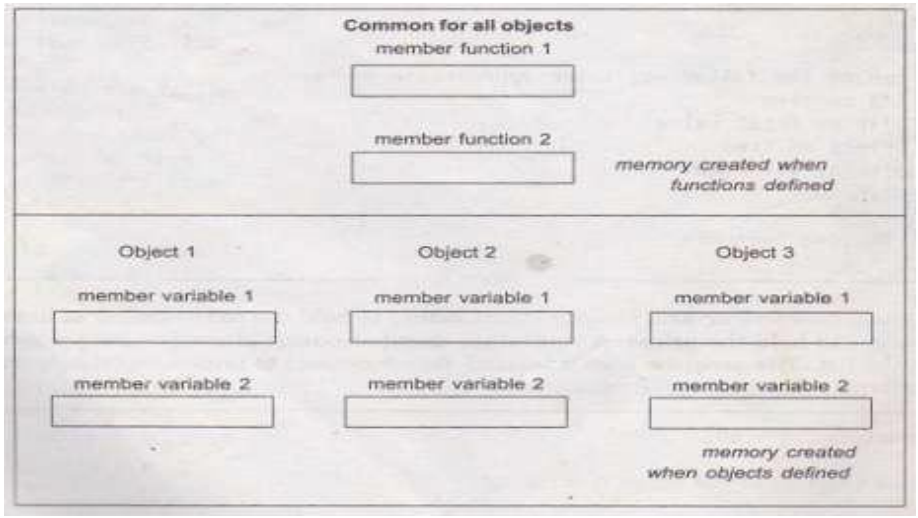
c) Ans. **Differentiate between constructor and destructor (any four points).** **4M**

Constructor	Destructor
A constructor is a special member function whose task is to initialize the objects of its class.	A destructor is a special member function whose task is to destroy the objects that have been created by constructor.
It constructs the values of data members of the class.	It does not construct the values for the data members of the class.
It is invoked automatically when the objects are created.	It is invoked implicitly by the compiler upon exit of a program/block/function.
Constructors are classified in various types such as : Default constructor Parameterized constructor Copy constructor Overloaded constructor	Destructors are not classified in any types.
A class can have more than one constructor.	A class can have at the most one constructor.
Constructor accepts parameters. Also it can have default value for its parameter.	Destructor never accepts any parameter.

Any four relevant points 1M each

		Syntax: classname() {… … }	**Syntax:** *destructor name is preceded with tilde.* ~classname() {…. …. }

		Example:
		ABC()
		{
		…
		}
			Example:
			~ABC()
			{
			….
			}

2.	**a)** **Ans.**	**Attempt any four of the following:** **Describe memory allocation for object with diagram.** The memory space for object is allocated when it is declared & not when the class is specified. The member functions are created &placed in memory space only once when they are defined as a part of a class definition. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions. Separate memory locations for the objects are essential because the (data) member variables will hold different data values for different objects.  	**16** **4M** ***Relevant Explanation 2M*** ***Correct Diagram 2M***

		In the above diagram, member functions 1 and 2 are stored in the common memory space as they require access by all objects. Each object (object 1, object 2, object 3) has its own separate memory space for its member variables.	

	b) Ans.	**Explain virtual function with suitable example.**
		Virtual Function:
		A virtual function is a member function that is declared within a base class and redefined by its derived class. When base class and its derived class both contain same name and prototype member function then derived class function overrides base class function. Base class pointer is used to refer member functions of its class as well as its derived class. When base pointer is used to refer to functions, it ignores the contents of the pointer and selects the member function that matches the function call. When both the classes contain same name and prototype function, base pointer executes a function from base class without considering the address inside the pointer. To execute derived class version of the overridden function virtual keyword is used with base class function. When a function is made virtual, compiler checks the address stored inside the pointer. If the pointer points to base class then function from base class is executed. If it contains address of derived class then function from derived class is executed. Run time polymorphism requires virtual function to execute same name function from base class and derived class depending on address stored inside the pointer.
		
		Example:
		
		#include<iostream.h>
		class Base
		{
		public:
		virtual void show( )
		{
		cout<<"\n show base";
		}
		};
		class Derived : public Base
		{
		public:
		void show( )
		{
			4M
			
			Correct Explanation 2M
			
			Correct Example 2M

```
cout<<"\n show derived";
}
};
void main( )
{
Base B,*bptr;
Derived D;
bptr=&B;
bptr->show( );
bptr=&D;
bptr->show( );
}
```

In above example, both base and derived class contains same name function as show. By creating a pointer of base class one can invoke desired show function by storing address of respective object in pointer.

c) Ans. **Explain different visibility modes and its effect in inheritance.** — **4M**

Different visibility modes are:

1) Private
2) Protected
3) Public

Visibility modes 1M

Effect in inheritance:

Base class visibility	Derived class visibility: Public derivation	Private derivation	Protected derivation
Private →	Not inherited	Not inherited	Not inherited
Protected →	Protected	Private	Protected
Public →	Public	Private	Protected

Private members of base class are not inherited directly in any visibility mode.

1) Private visibility mode

In this mode, protected and public members of base class become private members of derived class.

Relevant explanation of three modes 1M each

		2) Protected visibility mode In this mode, protected and public members of base class become protected members of derived class. **3) Public visibility mode** In this mode, protected members of base class become protected members of derived class and public members of base class become public members of derived class	
	d) **Ans.**	**List any six characteristics of OOP. Also list any two OOP languages.** **Characteristics of OOP:** 1) Emphasis is on data rather than procedure. 2) Programs are divided into objects. 3) Data structures are designed such that they characterize the objects. 4) Functions that operate on the data of an object are tied together in the data structure. 5) Data is hidden and cannot be accessed by external functions. 6) Objects may communicate with each other through functions. 7) New data and functions can be easily added whenever necessary. 8) It follows bottom-up approach in program design. **OOP Languages:** 1) Simula 2) Smalltalk 3) Objective C 4) C++ 5) Ada 6) Object Pascal 7) Turbo Pascal 8) Eiffel 9) Java	**4M** ***Any six characteristics 1/2M each*** ***Any two OOP languages ½ M each***
	e) **Ans.**	**Demonstrate the concept of friend function with example.** **Friend function:** The private members of a class cannot be accessed from outside the class but in some situations two classes may need access of each other's private data. So a common function can be declared which can be made friend of more than one class to access the private data of more than one class. The common function is made friendly with all those classes whose private data need to be shared in that function. This common function is called as friend function. Friend function is not in the scope	**4M** ***Explanation 2M***

		of the class in which it is declared. It is called without any object. The class members are accessed with the object name and dot membership operator inside the friend function. It accepts objects as arguments.
		Example:
		#include <iostream.h>
		#include<conio.h>
		class xyz;
		class abc
		{
		int a;
		public:
		void get1()
		{
		cin>>a;
		}
		friend void add(abc,xyz);
		};
		class xyz
		{
		int x;
		public:
		void get1()
		{
		cin>>x;
		}
		friend void add(abc,xyz);
		};
		void add(abc a1,xyz x1)
		{
		cout<<a1.a+x1.x;
		}
		void main()
		{
		abc a1;
		xyz x1;
		a1.get1();
		x1.get1();
		add(a1,x1);
		getch();
		}
			Any Example 2M

	f)
	
	Ans.
		Implement a program to declare a class city with data members city name and state. Accept and display data for 1 object using pointer to object.
		#include<iostream.h>
		#include<conio.h>
		class city
		{
		char city_name[20],state[20];
		public:
		void accept()
		{
		cout<<"\nEnter city data:";
		cout<<"\nName:";
		cin>>city_name;
		cout<<"\nState:";
		cin>>state;
		}
		void display()
		{
		cout<<"\nCity data is:";
		cout<<"\nName:"<<city_name;
		cout<<"\nState:"<<state;
		}
		};
		void main()
		{
		city c,*ptr;
		clrscr();
		ptr=&c;
		ptr->accept();
		ptr->display();
		getch();
		}
			4M
			
			class definition with functions 2M
			
			Main function with pointer to object concept 2M

3.	**a)** **Ans.**	**Attempt any four of the following:** **Explain memory management operator with example.** There are two memory management operators in C++: 1. new 2. delete	**16** **4M**

		These two memory management operators are used for allocating and de-allocating memory blocks. C++ allow dynamic allocation techniques when it is not known in advance how much of memory space is needed. **New operator:** The new operator in C++ is used for dynamic storage allocation. This operator can be used to create object of any type. **Syntax:** pointer variable = new datatype; In the above statement, new is a keyword and the pointer variable is a variable of type datatype. **Example:** 1. int *a = new int; 2. *a = 10; or 3. int *a = new int(10); In the above example, the new operator allocates sufficient memory to hold the object of data type int and returns a pointer to its starting point. The pointer variable holds the address of memory space allocated. **Delete operator:** The delete operator in C++ is used for releasing memory space when the object is no longer needed. Once a new operator is used, it is efficient to use the corresponding delete operator for release of memory. **Syntax:** delete pointer_variable; **Example:** Delete p;	***Explanation with example of new 2M*** ***Explanation with example of delete 2M***
	b) **Ans.**	**Explain the concept of constructor with default arguments with example.** C++ allows defining a constructor with default arguments. Programmer can declare a parameterized constructor in which a parameter (argument) can have default value. When a constructor with default argument is invoked it initializes data members either with default value or with value passed with function	***4M*** ***Explanation 2M***

		call. If function call does not contain value for default argument then default value is used to initialize data member but if value is pass for default argument then passed value is used for initialization.
		If a constructor requires three arguments and one of them is default argument then a function call to constructor requires only two values to execute.
		
		Example:
		class ABC
		{
		int a,b;
		public:
		ABC(int x,int y=2)
		{
		a=x;
		b=y;
		cout<<a<<b;
		}
		};
		
		void main()
		{
		ABC p(10);// first call to constructor
		}
		
		In the above example, constructor ABC has two arguments. One of the arguments has default value.
		In first call, constructor executes and it initializes variable with value 10 and b with default value 2.
		In second call, constructor executes and it initializes variable with value 10 and b with default value 20.
			Example 2M

	c) Ans.	**Explain constructor in derived class using one example.** When a class is declared, a constructor can be declared inside the class to initialize data members. When 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 arguments to the base class constructor. When both the derived and base classes contain constructors, the base constructor is executed first and then the constructor in the derived class is executed.	**4M** ***Explanation 2M***

The constructor of derived class receives the entire list of values as its arguments and passes them on to the base constructors in the order in which they are declared in the derived class.

General form to declare derived class constructor:
Derived-constructor (arglist1,arglist(D)):Base1(arglist1)
{
Body of derived class constructor
}
Derived constructor declaration contains two parts separated with colon (:). First part provides declaration of arguments that are passed to the derived constructor and second part lists the function calls to the base constructors.

Example:

Example 2M

```
#include<iostream.h>
#include<conio.h>
class base
{
int x;
public:
        base(int a)
        {
        x=a;
cout<<"Constructor in base. x="<<x;
        }
};
class derived: public base
{
int y;
        public:
        derived(int a,int b):base(a)
        {
         y=b;
cout<<"Constructor in derived.y="<<y;
        }
};
int main()
{
        clrscr();
```

		derived ob(2,3);
		getch();
		return 0;
		}
		In the above example, base class constructor requires one argument and derived class constructor requires one argument. Derived class constructor accepts two values and passes one value to base class constructor.	

	d)
	
	Ans.
		Use the concept of operator overloading to overload unary '-' operator to negate value of variables.
		Note: Any other correct logic shall be considered
		
		# include <iostream.h>
		#include<conio.h>
		class unary
		{
		int x, y, z;
		public:
		void getdata (int a, int , int c);
		void display (void);
		void operator - (); // overload unary minus.
		};
		void unary :: getdata (int a, int b, int c)
		{
		x = a;
		y = b;
		z = c;
		}
		void unary :: display (void)
		{
		cout<< x << " " << y << " " << z << "\n";
		}
		void unary ::operator - ()
		{
		x = -x ;
		y = -y ;
		z = -z ;
		}
			4M
			
			logic for operator overloading 3M
			
			Function Call 1M

		main ( )
		{
		clrscr();
		unary u;
		u.getdata(20, -30, 40);
		cout<< " u : " ;
		u. display ( ) ;
		-u;
		cout<< " u : " ;
		u. display ( ) ;
		}	

	e)
	Ans.
		Explain pointer to derived class with example.
		Pointers can be used to point to the base class objects and objects of derived class. Pointers to objects of base class are compatible with pointers to objects of a derived class. Single pointer variable can be made to point objects belonging to different classes.
		If B is base and D is derived class then pointer declared as a pointer to B can also be a pointer to D.
		Example:
		B *cptr; // pointer to of class
		B b; //Base object
		D d; // Derived object
		cptr=&b; //cptr store address of object b of base class
		cptr=&d; //cptr store address of object d of derived class
		
		It's not possible to access the public members of the derived class D by using cptr.Using base class pointer to object cptr, only those members inherited from B can be accessed and not the members that originally belong to D. For a member of D has the same name as any of the member of B, then reference to that member by cptr will always access the base class member. While C++ allows a base pointer to point to any object derived from that base, the pointer cannot be directly used to access all the members of the derived class.
		Example:
		#include<iostream.h>
		#include<conio.h>
		class base
		{
		int a;
			4M
			
			Explanation 2M

		public:
		void get()
		{
		cout<<"\nEnter a value for A";
		cin>>a;
		}
		void put()
		{
		cout<<"\nThe Value for A "<<a;
		}
		};
		class derived : public base
		{
		int a;
		public:
		void get()
		{
		cout<<"\nEnter a value for A";
		cin>>a;
		}
		void put()
		{
		cout<<"\nThe Value for A "<<a;
		}
		};
		void main()
		{
		base *bptr,b;
		derived d;
		clrscr();
		cout<<"\nPointing to the base class ";
		bptr=&b;
		bptr->get();
		bptr->put();
		cout<<"\nPointing to the derived class ";
		bptr=&d;
		bptr->get();
		bptr->put();
		getch();
		}
			Example 2M

f) **Differentiate between function definition inside and outside the class (any four points).** **4M**

Ans.

Sr. No	Inside function definition	Outside function definition
1	A member function of a class is defined inside the class.	A member function of a class is declared inside class and as defined outside the class.
2	The declaration is followed by the definition of a function inside the class definition.	After a member function is declared inside the class, it must be defined (outside the class) in the program.
3	The definition of member function inside the class is like normal function definition.	The definition of member function outside the class differs from normal function definition, as the function name in the function header is preceded by the class name and the scope resolution operator (: :).
4	No need of scope resolution operator.	The scope resolution operator informs the compiler what class the member belongs to.

5	The syntax for defining a member function inside the class is
	Return_type function name(parameter_list)
	{
	// Body of the member function
	}
		The syntax for defining a member function outside the class is
		Return_typeclass_name :: function_name (parameter_list)
		{
		// body of the member function
		}

6	**Example:**
	class item
	{
	int number;
	float cost;
	public:
	void putdata(void)
	{
	cout<< number <<endl;
		Example:
		class item
		{
		int number;
		float cost;
		public:
		void getdata (int a float b);//declaration
		};

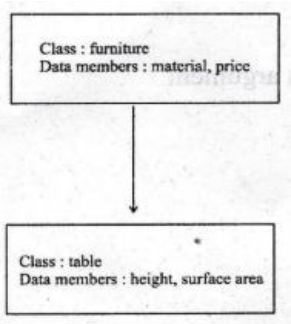
Any four relevant points 1M each

```
cout<< cost <<endl;
}
};
```

```
void item :: getdata(int a, float
b)
{
number = a;
cost = b;
}
```

4. **a)** **Attempt any four of the following:** **16**

Implement single inheritance for following fig. Accept and display data for 1 table. **4M**



Note: Any other correct logic shall consider

Ans.

```
#include<iostream.h>
class furniture
{
protected:
char material[20];
int price;
};
class table :public furniture
{
int height ;
float sur_area;
public:
void getdata()
{
cout<<"enter material";
```

Class furniture 1M

Class table 1M

accept and display data 2M

		cin>>material;
		cout<<"enter price";
		cin>>price;
		cout<<"enter height";
		cin>>height;
		cout<<"enter surface area";
		cin>>sur_area;
		}
		
		void putdata()
		{
		cout<<" material is"<<material<<endl;
		cout<<"price is"<<price<<endl;
		cout<<" height is "<<height<<endl;
		cout<<"surface area is "<<sur_area<<endl;
		}
		};
		void main()
		{
		table t1;
		t1.getdata();
		t1.putdata();
		}	

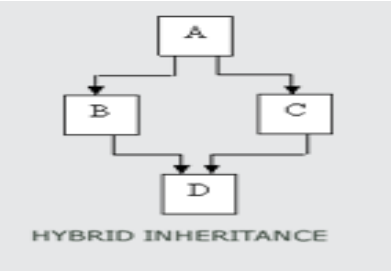
	b)
	
	Ans.
		Describe constructor with syntax and example.
		Note: Any constructor type shall be considered
		A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs the value data members of the class.
		
		Syntax:
		class_name( )
		{
		Constructor body
		}
		
		Example:
		class ABC
		{
			4M
			
			Explanation 2M
			
			Correct syntax 1M

		int a;
		public:
		ABC( )//constructor declaration
		{
		a=0;
		}
		};
		
		void main()
		{
		ABC x;
		}
			Example 1M

	c)
	Ans.
		Explain insertion and extraction operators in C++ with example.
		Insertion operator:
		The operator << is called as insertion operator works with cout to inserts the contents of the variable on screen (for output).
		Example:
		cout<<"Welcome to C++"; //Message is displayed on screen as it is.
		OR
		cout<<x;// Value of x will be printed on console / screen.
		
		Extraction operator:
		The operator >>is called as extraction operator or get from extracts the value from keyboard and assigns it to the variable on its right. Extraction operator is used with cin statement to accept input from user (keyboard).
		Example:
		cin>>number1;
			4M
			
			Explanation with example for insertion 2M
			
			For Extraction operator 2M

	d) **Ans.**	**List characteristics of static data member and static member function.** **Characteristics of static member variable are:** i) It is initialized to zero when the first object of its class is created. No other initialization is permitted. ii) Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created. iii) It is visible only within the class, but its lifetime is the entire program.	**4M** ***Characteristics of static data member 2M***

		iv) Static variables are normally used to maintain values common for all objects. **Characteristics of Static member function are:** i)A static member function can only have access to other static data members and functions declared in the same class. ii)A static member function can be called using the class name with a scope resolution operator instead of object name as follows: class_name::function_name;	***Static member function 2M***

	e) Ans.	**Explain hybrid inheritance with example.**
		Hybrid inheritance is also referred as mixed inheritances. As the name suggests it is a combination of all the kinds of inheritance mechanisms, namely single inheritance, multiple inheritance, multilevel inheritance and hierarchical inheritance.
		
		Example:
		 
		
		#include<iostream.h>
		class A
		{
		protected:
		int a;
		};
		class B:public virtual A
		{
		protected:
		int b;
		};
		class C:public virtual A
		{
		protected:
		int c;
		};
		class D:public B,public C
			4M
			
			Explanation 2M
			
			Example 2M

		{
		public:
		void getdata()
		{
		cin>>a>>b>>c;
		}
		void putdata()
		{
		cout<<a<<b<<c;
		}
		};
		void main()
		{
		D d;
		d.getdata();
		d.putdata();
		}	

	f)
	
	Ans.
		Write a program to search a number from an array using pointer to array.
		#include<iostream.h>
		#include<conio.h>
		void main()
		{
		int a[5],i,*a1,no,flag=1;
		clrscr();
		a1=&a[0];
		cout<<"\nEnter array elements :"<<endl;
		for(i=0;i<5;i++)
		{
		cout<<"\n\t Enter "<<i<<" element:";
		cin>>*a1;
		a1++;
		}
		cout<<"Enter element to be searched::";
		cin>>no;
		a1=&a[0];
		for(i=0;i<5;i++)
		{
		if(*a1==no)
		{
			4M
			
			Correct logic 2M
			
			Correct Syntax 2M

		cout<<"\n\t Number is present...";
		flag=1;
		break;
		}
		else
		{
		flag=0;
		a1++;
		}
		}
		if(flag==0)
		{
		cout<<"\n\t Number is not present.... ";
		}
		getch();
		}	

5.	**a)** **Ans.**	**Attempt any four of the following:** **Write any four rules for operator overloading.** **Rules for operator overloading:** 1. Only existing operators can be overloaded. New operators cannot be created. 2. The overloaded operator must have at least one operand that is of user defined type. 3. We cannot change the basic meaning of an operator i.e. we cannot redefine the plus(+) operator to subtract one value from the other. 4. Overloaded operators follow the syntax rules of the original operators. They cannot be overridden. 5. There are some operators that cannot be overloaded. for e.g. sizeof, . , .* , : : , ?: 6. We cannot use friend functions to overload certain operators (=,( ),[ ],->).However member functions can be used to overload them. 7. Unary operators overloaded by means of a member function, take no explicit arguments and return no explicit values, but those overloaded by means of a friend function, take one reference argument. 8. Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments. 9. When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.	**16** **4M** ***Any four rules 1M each***

		10. Binary arithmetic operators such as +,-,* and / must explicitly return a value. They must not attempt to change their own arguments.	

	b) Ans.	**Explain structure with syntax and example.**
		Structure:
		The Structure is a user defined data supported by object oriented programming.
		It has almost similar properties that any other user defined data type possess except all members are public by default.
		One can create a structure using following syntax:
		
		struct structure_name
		{
		data_member1;
		data_member2 ;
		.
		.
		data_memberN;
		};
		
		Example:
		
		#include<iostream.h>
		#include<conio.h>
		struct demo
		{
		int a;
		};
		void main()
		{
		demo d;
		clrscr();
		cout<<"\nEnter a value for demo's a";
		cin>>d.a;
		cout<<"\nThe Value is "<<d.a;
		getch();
		}
			4M
			
			Explanation with syntax 2M
			
			Example 2M

c) Ans. **Compare run-time and compile-time polymorphism.** **4M**

Sr. No	Run-time Polymorphism	Compile-time Polymorphism
1.	It simply means that selection of appropriate function is done at run time.	It simply means that an object is bound to its function call at compile time.
2.	Function to be called is unknown until appropriate selection is made.	Functions to be called are know well before.
3.	This requires use of pointers to object.	This does not require use of pointers to objects.
4.	Function call execution is slower.	Function calls are faster.
5.	Also called as late binding.	Also called as early binding.
6.	E.g. virtual function.	E.g. overloaded function call.
7.	It also referred as Dynamic Binding.	It also referred as Static Binding.

Any 4 Points of comparison 1M each

d) Ans. **Explain any four concept of OOP.** **4M**

Basic Concepts of Object Oriented Programming:

1. Objects

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. An object is the instance of the class. When a program is executed, the objects interact by sending messages to one another.

Any 4 Concepts 1M each

2. Classes

A class is the collection of related data and function under a single name. A class is collection of object of similar type. The entire set of data and code of an object can be made a user-defined data type with the help of class. Once a class has been defined, we can create any number of objects belonging to that class. Classes are user-defined that types and behave like the built-in types of a programming language.

3. Data Abstraction and Encapsulation

The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. This insulation of the data from direct access by the program is called

		data hiding or information hiding. Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction; they encapsulate all the essential properties of the object that are to be created. **4. Inheritance** Inheritance is the process by which objects of one class acquired the properties of objects of another classes. In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined feature of both the classes. **5. Polymorphism** Polymorphism means the ability to take more than on form. An operation may exhibit different behavior is different instances. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. **6. Dynamic Binding** Binding refers to the linking of a procedure call to the code to be executed in response to the call. **7. Message Passing** An object-oriented program consists of a set of objects that communicate with each other. Objects communicate with one another by sending and receiving information.	
	e) **Ans.**	**Describe pointer arithmetic with example.** 1. As a pointer holds the memory address of a variable some arithmetic operations can be performed with pointers. C++ supports four arithmetic operators that can be used with pointer such as increment++ 2. Pointers are variables. They are not integers, but they can be displayed as unsigned integers. The conversion specifier for a pointer is added and subtracted. For example: Ptr++: causes the pointer position to be incremented, but not by 1. Ptr --: the pointer position to be decremented, but not by 1.	**4M** ***Description 2M***

		3. Following program segment shows the pointer arithmetic.
		The integer value would occupy bytes 2000 and 2001
		int value, * ptr; value=120;
		ptr=&value;
		ptr ++;
		cout<<ptr;
		
		Example:
		
		#include<iostream.h>
		#include<conio.h>
		void main()
		{
		int value;
		int*ptr;
		ptr= &value;
		cout<<"memory address before increment=";
		cout<<ptr<<endl; ptr++;
		cout<<" memory address after increment=";
		cout<<ptr<<endl;
		}
		memory address before increment=0X24c8ff4
		memory address after increment=0X24c8ff5
			Example 2M

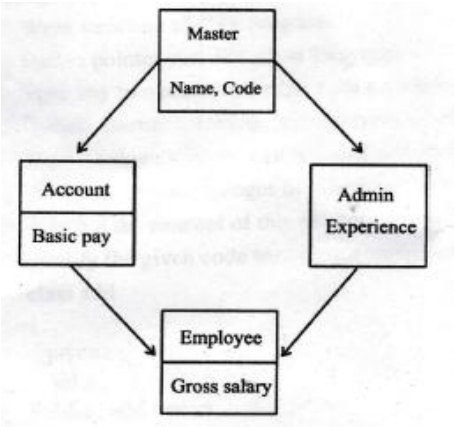
	f)
	
	Ans.
		Write a program to calculate area of circle and area of rectangle using function overloading.
		
		#include<iostream.h>
		#include<conio.h>
		float area(float a)
		{
		return (3.14*a*a);
		}
		int area(int p,int q)
		{
		return(p*q);
		}
		void main()
		{
		clrscr();
			4M
			
			Each area function 1M calling function with valid argument 1M
			
			Main function 1M

		cout<<"Area of circle:"<<area(6);
		cout<<"Area of Rectangle:"<<area(5,6);
		getch();
		}	

6.	**a)**
	
	Ans.
		Attempt any two of the following:
		Write a program to declare a class 'staff' having data members as name and department. Accept this data for 10 staffs and display names of staff that are in 'CO' department.
		#include<iostream.h>
		#include<conio.h>
		#include<string.h>
		class staff
		{
		char name[10], dept[10];
		public:
		void accept()
		{
		cout<<"Enter Name and Department:\t";
		cin>>name>>dept;
		}
		void display()
		{
		if(strcmp(dept,"CO")==0|| strcmp(dept,"co")==0)
		{
		cout<<"\nStaff name::\t"<<name<<"\t"<<"Department is ::"<<dept;
		}
		}
		};
		void main()
		{
		staff s[10];
		int i;
		clrscr();
		for(i=0;i<=10;i++)
		{
		s[i].accept();
		}
		for(i=0;i<=10;i++)
			16
			8M
			
			Defining class with specifications 4M
			
			Displaying values for condition 2M
			
			Creating 10 Objects 1M
			
			Calling Functions 1M

```
{
 s[i].display();
}
getch();
}
```

b) **Write a program to implement the concept of virtual base class for following figure. Accept and display information of one employee with his name, code, basic pay, experience and gross salary with the object of employee class.** **8M**



Ans.

```
#include<iostream.h>
#include<conio.h>
class Master
  {
     char name[10],code[3];
   public:
     void acceptM()
          {
          cout<<"\nEnter name and code ";
          cin>>name>>code;
          }
     void displayM()
          {
          cout<<"\nThe name and code is"<<name<<code;
          }
  };
class Account : public virtual Master
```

Each Class 1 ½ M

Main function 2M

```
  {
     float basic_pay;
   public:
     void virtual acceptA()
          {
          cout<<"\nEnter Basic Pay ";
          cin>>basic_pay;
          }
     void virtual displayA()
          {
          cout<<"\nThe Basic Pay is"<<basic_pay;
          }
  };
class Admin : public virtual Master
  {
     float experience;
   public:
     void virtual acceptD()
          {
          cout<<"\nEnter Experience ";
          cin>>experience;
          }
     void virtual displayD()
          {
          cout<<"\nThe Experience is"<<experience;
          }
  };
class Employee : public Admin, public Account
  {
     float gross_sal,da;
   public:
     void acceptE()
          {
          cout<<"\nEnter Gross Salary ";
          cin>>gross_sal;
          }
     void displayE()
          {
          cout<<"\nThe Gross Salary is "<<gross_sal;
          }
```

		`};`
		`void main()`
		`{`
		`Employee e;`
		`clrscr();`
		`e.acceptM();`
		`e.acceptA();`
		`e.acceptD();`
		`e.acceptE();`
		`e.displayM();`
		`e.displayA();`
		`e.displayD();`
		`e.displayE();`
		`getch();`
		`}`	

	c)
	
	Ans.
		Write a program to find length of a string using pointer to string.
		
		`#include<iostream.h>`
		`#include<conio.h>`
		`void main()`
		`{`
		`char str1[10],*ptr;`
		`int len=0;`
		`cout<<"enter string:";`
		`cin>>str1;`
		`ptr=&str1[0];`
		`while(*ptr!='\0')`
		`{`
		`len++;`
		`ptr++;`
		`}`
		`cout<<"\nThe Length of a string is"<<len;`
		`getch();`
		`}`
			8M
			
			Pointer creation and accepting string 2M
			
			Pointer to initial position 1M
			
			Calculation of Length 4M
			Display length 1M