WINTER – 2019 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.

Q. No.	Sub Q.N.	Answer	Marking Scheme
1.	**(A)** **(a)** **Ans.**	**Attempt any SIX of the following:** **State any four applications of OPP.** **Applications of OOP:** • Real time systems • Simulation and modeling • Object-oriented databases • Hypertext, hypermedia and expert Ext • AI and expert systems • Neural networks and parallel programming • Decision support and office automation systems • CIM/CAM/CAD systems	**12** **2M** ***Any four correct applications ½M each***
	(b) **Ans.**	**Define pointer.** Pointer is a variable which holds an address of another variable of same data type.	**2M** ***Correct definition 2M***
	(c)	**Give output for the following code:** **class student** **{**	**2M**

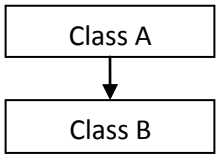
	Ans.	**int roll no;** **float per;** **char name[14];** **} S[5];** **void main ( )** **{** **cout<< sizeof (S);** **}** **Output:** Since in above code variable name roll no is an invalid variable therefore output will be as invalid variable declaration **OR** OUTPUT= 100	***Correct output 2M***
	(d) Ans.	**State any four types of constructor.** **Types of constructor:** • Default constructor • Parameterized Constructor • Constructor with Default Argument • Copy Constructor • Multiple/overloaded constructor	**2M** ***Any four types ½M each***
	(e) Ans.	**Write any two rules for virtual function.** **Rules for virtual function:** • A virtual function must be member of some class. • Virtual functions cannot be static. • Virtual functions can be a friend of another class. • Virtual functions should be accessed using pointers. • A virtual function in based class must be defined, even though it may not be used. • The prototype of virtual functions should be same in base as well as derived class. • If it is defined in base class, it need not be necessarily redefined in derived class. • A class may have virtual destructor but it cannot have a virtual constructor. • While a base pointer can point to any type of derived object, the reverse is not true. • When a base pointer points to a derived class, incrementing or decrementing it will not make it to point to the next object of	**2M** ***Any two rules 1M each***

		derived class.	
	(f) **Ans.**	**Define abstract class.** An abstract class is a class that is designed only to act as base class. It is not used to create objects.	**2M** ***Correct Definition 2M***
	(g) **Ans.**	**State two advantages of pointer.** **Advantages of pointer:** • Pointers save the memory. • Pointers reduce the length and complexity of a program. • Pointers allow passing of arrays and strings to functions more efficiently. • Pointers make possible to return more than one value from the function. • Pointers increase the processing speed.	**2M** ***Any two advantages 1M each***
	(h)	**Give output for the following code:**	**2M**

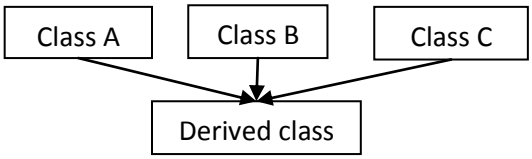
```
class ABC
{
int x;
public;
ABC( )
{
  Cout << “Welcome”;
}
ABC (int y)
{
  x = y;
  cout << x;
}
};
void main ( )
{
  ABC b,c (10);
}
```

	Ans.	Output: Welcome 10.	***Correct Output 2M***
1.	**(B)** **(a)**	**Attempt any TWO of the following:** **Describe the concept of constructor with default argument with suitable example.**	**8** **4M**

	Ans.	**Definition**: The constructor where we can assign default values for one or more parameters at the time of function declaration is called as constructor with default argument
		
		Example:
		class complex
		{
		float real, img ;
		public:
		complex (float r,floati=0.0)
		{
		real=r;
		img=I;
		}
		void display()
		{
		cout<< real<<"+"<<img<<"i"<<endl;
		}
		}
		void main()
		{
		complex c1(5.0);
		complex c2(2.0,3.0);
		cout<<"complex c1:\n";
		c1.display();
		cout<<"complex c2:\n";
		c2.display();
		getch();
		}
		
		These corresponding parameters are omitted in the call to the constructor
		For example the constructor complex() can be declared as follows:
		complex (float real, float img=0.0);
		Here, if default value of the argument img is set to zero, then the statement:
		complex c(5.0); Assigns the value 5.0 to real and 0.0 to img (by default).
		complex c(2.0,3.0); Assigns 2.0 to real and 3.0 to img., because the actual parameters, when specified overrides the default value.
			Explanation 2M
			
			Example 2M

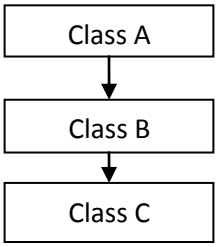
	(b) Ans.	**State and describe types of inheritance.**
		Types of inheritance:
		○ Single Inheritance
		○ Multiple Inheritance
		○ Multilevel Inheritance.
		○ Hierarchical Inheritance
		○ Hybrid Inheritance
		
		• **Single Inheritance**
		○ The mechanism of deriving a new class from existing single base class is known as Single Inheritance.
		○ Syntax:
		class A
		{
		// Class A Body
		};
		class B: public class A
		{
		// class B Body;
		};
		 
		
		• **Multiple Inheritance**
		○ The mechanism of deriving a new class from several base classes is known as multiple Inheritance.
		○ Syntax:
		class A
		{
		// Class A Body
		};
		class B:
		{
		// class B Body;
		};
		class C
			4M
			
			Any four types 1M each

```
{
 // Class C Body
};
class Derived: public A, public B, public C
{
 // class Derived Body;
};
```



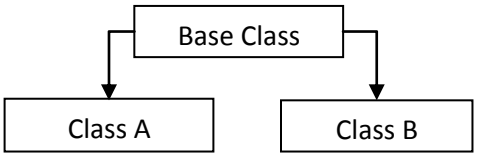
- **Multilevel Inheritance**
  - The mechanism of deriving a new class from already existing derived class is known as Multilevel Inheritance.
  - In this type of Inheritance a child class can access properties of its parent class as well as grandparent class
  - Syntax:

```
class A
{
 // Class A Body
};
class B: public A
{
 // class B Body;
};
class C: public B
{
 // Class C Body
};
```



- **Hierarchical Inheritance**
  - The mechanism of deriving multiple derived classes from a single base class is known as Hierarchical Inheritance.
  - In this inheritance all derived classes can access properties of ingle base class.
  - Syntax:

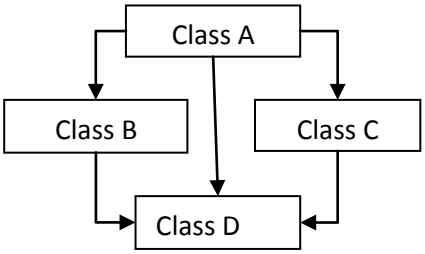
```
class Base
{
 // Class Base Body
};
class A: public Base
{
 // class A Body;
};
class B: public Base
{
 // Class C Body;
};
```



- **Hybrid Inheritance**
  - A design of inheritance consisting more than one type of Inheritances is known as Hybrid Inheritance
  - Syntax

```
class A
{
 // Class A Body
};
class B: virtual public A
{
 // class B Body;
};
class C: public virtual A
{
 // Class C Body;
```

```
};
class D: public B, public C
{
 // class D body;
};
```



(c) **Ans.**

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

Constructor	Destructor
1. Constructor is used to initialize the object of its class	1. Destructor is used to destroy the objects that have been created by a constructor.
2. It has same name as the class name.	2. It has same name as the class name but is preceded by tilde sign (~).
3. It can be invoked automatically when the object of its class is created.	3. It is invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible.
4. It can have arguments.	4. It doesn't take any arguments

5. Syntax:
class_name
(arg1,arg2,…,argn)
{
Body of constructor;
}
	5. Syntax:
	~class_name
	(arg1,arg2,…,argn)
	{
	Body of destructor;
	}

Any four correct points 1M each

WINTER – 2019 EXAMINATION
MODEL ANSWER

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

2.		**Attempt any FOUR of the following:**	**16**
	(a)	**State and describe access specifiers used inside class to declare members.**	**4M**
	Ans.	**Access specifiers:** 1. private 2. protected 3. public	***List 1M***
		Private access specifier: Class members declared as private can be accessed only from within the class. Outside class access is not allowed for private members of class. By default members are private. **Protected access specifier:** Class members declared as protected can be accessed by the member functions within its class and any class immediately derived from it. These members cannot be accessed by the functions outside these two classes. **Public access specifier:** Class members declared as public can be accessed from outside the class also.	***Description 1M each***

		Example:-
		class base
		{
		private:
		int a;
		protected:
		int b;
		public:
		void display()
		{
		cout<<a<<b;
		}
		};
		class derived:public base
		{
		public:
		void show()
		{
		cout<<b;
		}	

		`};`
		`void main()`
		`{`
		`derived d;`
		`d.display();`
		`d.show();`
		`}`
		
		• In the above example, variable 'a' can be access by its member function display () as it is a private variable.
		• Variable 'b' can be accessed by its member function display () as well as member function show () of its derived class as it is a protected member.
		• Member function display () and show () can be accessed from main () as they are public members of class	

	(b) **Ans.**	**With suitable example, describe use of virtual function in polymorphism.** • In order to achieve polymorphism, objects belonging to different classes should be able to respond to the same message at different instances which initiates the use of single pointer variable to refer to objects of different classes. • In case of inheritance, base class pointer is used to refer to all derived objects but even when it contains an address of derived class always executes base class function. • The Compiler simply ignores content of pointer & selects member function that matches type of pointer. In order to achieve polymorphism, the concept of virtual functions is used. • When we use same function name in both base & derived classes, function in **base class** is declared as **virtual** using **keyword virtual** preceding its normal declaration. • When function is made virtual C++ determines which function to be executed at runtime based on type of object pointed to by base pointer rather than type of pointer. • Thus by making use of single pointer variable (base pointer) to point to different objects so that different version of virtual function can be executed and hence polymorphism is achieved known as **Run Time Polymorphism.**	**4M** ***Description 2M***

		Example:
		#include<conio.h>
		#include<iostream.h>
		class student
		{
		int roll;
		char name[20];
		public:
		virtual void accept()
		{
		cout<<"\n Enter Roll id:";
		cin>>roll;
		cout<<"\n Enter name:";
		cin>>name;
		}
		virtual void display()
		{
		cout<<"\n Roll id:"<<roll<<endl;
		cout<<"\n Name:"<<name<<endl;
		}
		};
		class Test: public student
		{
		int marks1,marks2;
		public:
		void accept()
		{
		student::accept();
		cout<<"\n Enter marks 1:";
		cin>>marks1;
		cout<<"\n Enter marks 2:";
		cin>>marks2;
		}
		void display()
		{
		student:: display();
		cout<<"Marks1="<<marks1<<endl;
		cout<<"Marks2="<<marks2<<endl;
		}
			Example 2M

```
};
void main()
{
student *p;
 Test b;
 p=&b;
clrscr();
p->accept();
p->display();
getch();
}
```

(c) Ans. **State and describe visibility modes used in inheritance with their effects.** — **4M**

Visibility modes: — ***List 1M***

- private
- protected
- public

Base class visibility	**Derived class visibility**		
	Private	Protected	Public
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

- **Private:**
  - When a base class is privately inherited by a derived class, 'public members' and 'protected members' of the base class become 'private members' of the derived class.
  - Therefore, the public and protected members of the base class can only be accessed by the member functions of derived class but, cannot be accessed by the objects of the derived class.

Syntax:

```
class derived: private base
{
//Members of derived class;
 };
```

- **Public:**
  - When a base class is publicly inherited by a derived class then

Description of each 1M

		'protected members' of base class becomes 'protected members' and 'public members' of the base class become 'public members' of the derived class.
		o Therefore the public members of the base class can be accessed by both the member functions of derived class as well as the objects of the derived class.
		Syntax:
		class derived: public base
		{
		//Members of derived class;
		};
		
		• **Protected:**
		o When a base class is protectedly inherited by a derived class, 'public and protected members' of the base class become 'protected members' of the derived class.
		o Therefore the public and protected members of the base class can be accessed by the member functions of derived class as well as the member functions of immediate derived class of it but they cannot be accessed by the objects of derived class
		Syntax:
		class derived: protected base
		{
		//Members of derived class;
		};	

	(d) **Ans.**	**Define the following terms:** **(i) Data abstraction** **(ii) Class** **(iii) Dynamic binding** **(iv) Polymorphism** **(i) Data abstraction:** Abstraction refers to the act of representing essential features without including the background details or explanation. Data abstraction is the process of defining a data type, often called abstract data type (ADT), together with the principle of data hiding. **(ii) Class:** Class is a collection of objects of same data types. Class contents data	**4M**

		and functions that operate on data. **(iii) Dynamic binding:** Linking of function call to function definition at run time is defined as dynamic binding **(iv) Polymorphism:** Polymorphism means ability to take more than one form at different instances depending on the type or number of arguments.	***Correct Definition of each term 1M***

	(e) Ans.	**With example, describe use of static member function.**
		
		• A static member function can have access to only other static members (functions or variables) declared in the same class.
		• A static member function can be called using the class name as follows:
		• class_name::function_name;
		• A static member of a class does not depend on any specific object of class. A static member function can be called without existence of any of the class object.
		• So static member function is used to access static members without any specific object of class.
		
		Example:-
		class test
		{
		static int count;----------//static data memeber
		public:
		void setcount() ----------//member function
		{
		count=count+1;
		}
		static void showcount()----------//static member function
		{
		cout<<count;
		}
		};
		int test::count;----------// static member declaration outside class
		void main()
			4M
			
			Explanation 2M
			
			Example 2M

		{
		test t1,t2;
		t1.setcount();
		t2.setcount();
		test::showcount(); ----------//Call to static member function.
		}	

	(f)
	
	Ans.
		Write a program in C++ to search an element from an array using pointer.
		(Note: Any other logic shall be considered)
		#include<iostream.h>
		#include<conio.h>
		void main()
		{
		int a[10], n, i,*p, flag=0, x;
		clrscr();
		cout<<"Enter no. of array elements \n";
		cin>>n;
		cout<<"Enter the array elements \n";
		for(i=0;i<n;i++)
		{
		cin>>a[i];
		}
		cout<<"Enter the key element \n";
		cin>>x;
		p=a;
		for(i=0;i<n;i++)
		{
		if(*(p+i)==x)
		{
		flag=1;
		cout<<x<<"is found at location"<<i+1<<endl;
		break;
		}
		}
		if(flag==0)
		cout<<x<<"is not found \n";
		getch();
		}
			4M
			
			Correct logic 2M
			
			Correct syntax 2M

WINTER – 2019 EXAMINATION
MODEL ANSWER

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

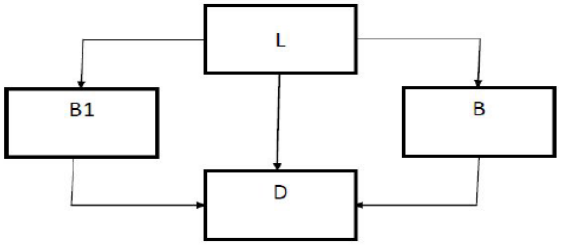
3.		**Attempt any FOUR of the following:**	**16**
	(a)	**With example, describe use of scope resolution operator.**	**4M**

	Ans.	In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator:: called scope resolution operator. This can be used to uncover a hidden variable. It takes the following form:
		:: variable;
		This operator allows access to the global version of a variable.
		Example:
		int student :: roll_no; // Using data members with the help of scope resolution operator
		or
		void student :: performance ( ) // Using member function with the help of scope resolution
		operator
		{ //Function Body }
		
		Example:
		# include<iostream.h >
		int m = 10;
		int main ( )
		{
		int m = 20;
		{
		int k = m;
		int m = 30;
		cout << “K =”<<k;
		cout << “m =”<<m;
		cout << “: : m =”<< : : m;
		}
		cout << “m=”<< m;
		cout << “::m =” <<::m;
		}
		return 0;
		}
			Use 2M
			
			Example 2M

	(b)	**Write a program in C++ to declare a class measure having data members as add 1, add 2, add 3. Initialize the values of two data members using constructor and display their addition using function.**	**4M**

	Ans.	***(Note: Any other logic shall be considered)***
		#include<iostream.h>
		#include<conio.h>
		class measure
		{
		public:
		int add1,add2,add3;
		measure(int a,int b)
		{
		add1=a;
		add2=b;
		}
		void cal()
		{
		add3=add1+add2;
		}
		void display()
		{
		cout<<"\n Sum="<<add3;
		}
		};
		void main()
		{
		int a, b;
		clrscr();
		cout<<"Enter a and b:";
		cin>>a>>b;
		measure m1(a, b);
		m1.cal();
		m1.display();
		getch();
		}
			Declaration of class measure 1M
			
			Initialization constructor 1M
			
			Display 1M
			
			Main ( )1M

	(c) Ans.	**Describe with example importance of virtual base class.** A virtual base class (Grandparent class) is a class that avoids duplication of inherited data in derived class (child class) derived from parent classes (parent1 and parent2) which in turn derived from base class.	**4M** ***Description 2M***

		
		
		w.r.t. above diagram:
		class L
		{
		/* ... */
		};
		// indirect base class
		class B1 : virtual public L { /* ... */ };
		class B : virtual public L { /* ... */ };
		class D : public B1, public B { /* ... */ };
		
		Two derived classes as shown in above diagram B1 and B have a common base class L, and another class D inherits properties of B1 and B base classes.
		Then class D inherits properties of class L two times from two base classes B1 and B. This leads to duplication of data in derived class D. To avoid this duplications class L is declare as virtual base class while defining base classes B1 and B.
			Example 2M

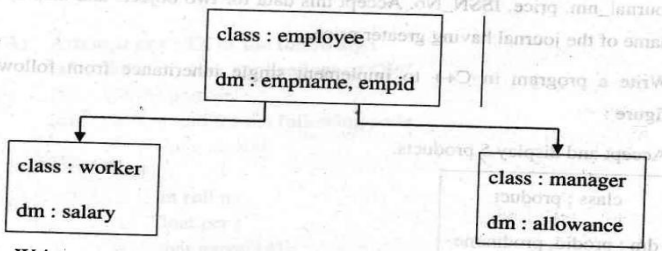
	(d) **Ans.**	**Write a C++ program to overload = = operator to check equality of two strings.** ***(Note: Any other logic shall be considered)*** #include<iostream.h> #include<conio.h> #include<string.h> class string { char str1[20]; public: void get() { cout<<"Enter string:";	**4M** ***Correct logic 2M*** ***Correct syntax 2M***

		cin>>str1;
		}
		void operator = =(string s1)
		{
		int x;
		x=strcmp(s1.str1,str1);
		if(x= =0)
		cout<<"Equal";
		else
		cout<<"Not equal";
		}
		};
		void main()
		{
		clrscr();
		string s,s1;
		s.get();
		s1.get();
		s==s1;
		getch();
		}	

	(e) **Ans.**	**With suitable example, describe use of this pointer.** 1. C++ uses a unique keyword called "this" to represent an object that invokes a member function. 2. This unique pointer is automatically passed to a member function when it is invoked. 3. "this" is a pointer that always point to the object for which the member function was called . 4. For example, the function call A.max () will set the pointer "this" to the address of the object A. Next time suppose we call B.max(), the pointer "this" will store address of object B. Consider the following ***example:*** #include<conio.h> #include<iostream> class sample { int a;	**4M** ***Use 2M*** ***Example 2M***

		public:
		void setdata(int x)
		{
		this ->a=x;
		}
		void putdata()
		{
		cout<<this ->a;
		}
		};
		void main()
		{
		clrscr();
		sample s;
		s.setdata(100);
		s.putdata();
		getch();
		}	

	(f)
	
	Ans.
		Describe syntax and use of defining member function outside class. Give one example.
		Member function that is declared inside a class has to be defined separately outside the class. These member functions associate a membership identify label in the header. This label tells the compiler which class the function belongs to.
		
		Syntax of a member function definition is.
		Return type class name:: function-name(argument declaration)
		{
		Function body
		}
		
		The membership label class-name:: tells the compiler that the function function-name belongs to the class-name. The symbol :: is called as scope resolution operator.
		
		Ex- the function getdata is coded as
		void item:: getdata(int a, float b)
		{
		number = a;
		cost = b;
			4M
			
			Description / Use 2M
			
			Syntax 1M
			
			Example 1M

		} The member function have some special characteristics :- i) Several different classes can use the same function name the membership label will resolve their scope. ii) Member function can access the private data of the class. A non member function cannot do so. iii) A member function can call another member function directly, without using the dot operator.	
4.	**(a)**	**Attempt any FOUR of the following:** **Write a program in C++ to implement following inheritance. Assume suitable data.**  ***(Note: Any other logic shall be considered)***	**16** **4M**

	Ans.	#include<iostream.h>
		#include<conio.h>
		class employee
		{
		int empid;
		char empname[20];
		public:
		void accept()
		{
		cout<<"\n enter empid, empname:"<<endl;
		cin>>empid>>empname;
		}
		void dis()
		{
		cout<<"\n empid"<<empid;
		cout<<"\n empname--"<<empname<<endl;
		}
		};
			Definition of employee 1M

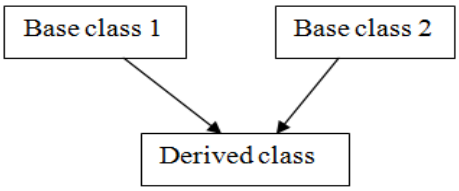
		class manager:public employee
		{
		int allowance;
		public:
		acc1()
		{
		accept();
		cout<<"\n Enter allowance:";
		cin>>allowance;
		}
		dis1()
		{
		dis();
		cout<<"\n Allowance:"<<allowance;
		}
		};
			Definition of manager 1M

		class worker:public employee
		{
		int salary;
		public:
		acc2()
		{
		accept();
		cout<<"\n enter salary";
		cin>>salary;
		}
		dis2()
		{
		dis();
		cout<<"\nSalary:"<<salary;
		}
		};
			Definition of worker 1M

		void main()
		{
		clrscr();
		manager m;
		worker w;
		m.acc1();
		m.dis1();
			Main() function 1M

		w.acc2();
		w.dis2();
		getch();
		}	

	(b) **Ans.**	**Write any four characteristics of constructor.** Rules to define constructor: 1. Constructors should be declared in the public section. 2. They are invoked automatically when the objects are created. 3. They do not have return type, not even void and therefore they cannot return values. 4. They can accept arguments. 5. They cannot be inherited, though derived class can call the base class constructor. 6. They cannot be virtual. 7. One cannot refer to their addresses. 8. An object with a constructor cannot be used as a member of a union. 9. They make implicit calls to the operators new and delete when memory allocation is required.	**4M** ***Any four characteristics 1M each***
	(c) **Ans.**	**Describe structure of C++ program.** General C++ program has following structure. INCLUDE HEADER FILES DECLARE CLASS DEFINE MEMBER FUNCTIONS DEFINE MAIN FUNCTION **Description:-** **1. Include header files** In this section a programmer include all header files which are require to execute given program. The most important file is *iostream.h* header file. This file defines most of the C++ statements like *cout* and *cin* . Without this file one cannot load C++ program. **2. Declare Class** In this section a programmer declares all classes which are necessary for given program. The programmer uses general syntax of creating class. **3. Define Member Functions** This section allows programmer to design member functions of a class. The programmer can have inside declaration of a function or	**4M** ***Description of structure 4M***

		outside declaration of a function. **4. Define Main Functions** This section the programmer creates object and call various functions writer within various class.	
	(d) **Ans.**	**State any four characteristics of friend function.** **Characteristics of friend function:** 1. It is not in the scope of the class to which it has been declared as friend. 2. As it is not in the scope of the class, it cannot be called using the object of that class. 3. It can be invoked like a normal function without the help of any object. 4. It cannot access the member names directly and has to use an object name and dot membership operator with each member name. 5. It can be declared either in the public or the private part of a class without affecting its meaning. 6. It has the objects as arguments.	**4M** ***Any four characteristics 1M each***
	(e) **Ans.**	**With example, describe multiple inheritance.** A derived class with multiple base classes is called as multiple inheritance. A derived class inherits properties of all the base classes. It also can have its own properties. *Syntax:-*  class base _class _name1 { ____ } ; class base_class _name2 { ____ } ; class derived_class_name :visibility_mode base_class_name_1 ,…, visibility mode base _class _name_n { ____ } ;	**4M** ***Description 2M***

		Example:
		#include<conio.h>
		#include<iostream.h>
		class base1
		{
		public: int b1;
		void get()
		{
		cout<<"\n Enter a number";
		cin>>b1;
		}
		void put()
		{
		cout<<"\n b=="<<b1;
		}
		};
		class base2
		{ public:
		int b2;
		void get1()
		{
		cout<<"\n Enter a number";
		cin>>b2;
		}
		void put1()
		{
		cout<<"\n b=="<<b2;
		} };
		class derived : public base1,public base2
		{
		public:
		void get2()
		{
		get();
		get1();
		}
		void put2()
		{
		put();
			Any correct example 2M

		put1();
		}
		};
		void main()
		{
		clrscr();
		derived d;
		d.get2();
		d.put2();
		getch();
		}	

	(f)
	Ans.
		Describe pointer to object with an example.
		When address of an object of a class is stored into the pointer variable of the same class type then it is pointer to object. This pointer can be used to access the data member and member functions of same class.
		
		Following example illustrate use of pointer to object
		#include<conio.h>
		#include<iostream.h>
		class product
		{
		private:
		int code;
		float price;
		public:
		void getdata(void)
		{
		cout<<"\n Enter code";
		cin>>code;
		cout<<"\n Enter price";
		cin>>price;
		}
		void display(void)
		{
		cout<<"\n Code="<<code<<"\n Price="<<price;
		}
		};
		void main()
		{
			4M
			
			Description 2M
			
			Example 2M

		product p1; //create object of product
		product *ptr; //create pointer of type product
		ptr=&p1; //ptr points to object p1
		ptr->getdata (); // Invoking getdata()using pointer to object p1.
		ptr->display();
		}	

5.	**(a)** **Ans.**	**Attempt any FOUR of the following:** **State 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. 10. Binary arithmetic operators such as +,-,* and / must explicitly return a value. They must not attempt to change their own arguments.	**16** **4M** ***Any four rules 1M each***
	(b) **Ans.**	**Differentiate between class and structure. (any four points)** **Class** | **Structure** 1. Class is user defined data type. It's a way of binding data and functions together in | 1. Structure contains logically related data items which can be of similar type or different	**4M**

		one single unit. It is a collection of data members and member functions. | type.	***Any four points 1M each***
		2. It allows data and functions to be hidden from external use. | 2. In structure data is not hidden from external use.	
		3. In class all members are by default are private. | 3. In structure all members by default are public.	
		4. In class object is created. | 4. In structure structure_variable is created.	
		5. Syntax: class class_name { access specifier: declare data members; declare member functions; }; | 5. Syntax: structstructure_name { datatype variable1; datatype variable2; .. .. }structure_variable;	

		for e.g.
		class student
		{
		private:
		introll_no;
		char name[20];
		public:
		void getdata();
		void putdata();
		}; | for e.g.
		struct student
		{
		introll_no;
		char name[20];
		}s;	

	(c)	**Write a program in C++ to overload a 'volume' function to calculate volume of cube and rectangular box.** ***(Note: Any other logic shall be considered.)***	**4M**

	Ans.	#include<iostream.h>
		#include<conio.h>
		void volume(float);
		void volume(float, float, float);
		void main()
		{
			Correct volume() for cube 1M

		float a, length, width, height;
		clrscr();
		cout<<"\n Enter side of a cube:";
		cin>>a;
		cout<<"\n Enter length, width and height of a rectangle:";
		cin>>length>>width>>height;
		volume(a);
		volume(length, width, height);
		getch();
		}
		void volume(float a)
		{
		float v=a*a*a;
		cout<<"\n Volume of a cube is:"<<v;
		}
		void volume(float length, float width, float height)
		{
		float v=length*width*height;
		cout<<"\n Volume of a rectangular box is:"<<v;
		}
			Correct volume() for rectangular cube 1M
			
			Correct main() with function calls 2M

	(d)
	
	Ans.
		Describe with example the use of insertion and extraction operators.
		
		Insertion operator:
		The operator "<<" is called as insertion operator. It 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
			
			Each description 1M
			
			Each example 1M

WINTER – 2019 EXAMINATION
MODEL ANSWER

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

(e) **State and describe use of pointer operator and address operator. Give one example of each.** **4M**

Ans.

Pointer operator:- *

It is used to declare a pointer variable. Also used as "value at" operator to read value stored inside the address pointed by pointer.

Example:

```
int *ptr;
```

Each description 1M

Address operator:-&

It is used to retrieve address of a variable. With address operator address of a variable can be stored in pointer variable.

Example:

```
int a,*ptr;
ptr=&a;
```

Each example 1M

(f) **Write any four differences between compile time and run time polymorphism.** **4M**

Ans.

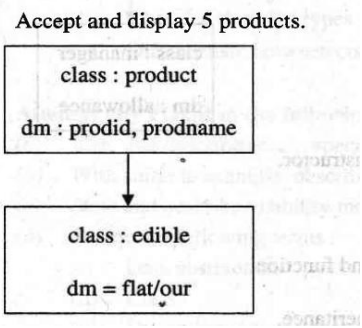
Sr. No.	Compile-time Polymorphism	Run-time Polymorphism
1.	Compile time polymorphism means that an object is bound to its function call at compile time.	Run time polymorphism means that selection of appropriate function is done at run time.
2.	Functions to be called are know well before.	Function to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

Any four differences 1M each

6.		**Attempt any TWO of the following:**	**16**
	(a)	**Write a program in C++ to declare a class 'Journal' having data members as journal_nm, price, ISSN_No. Accept this data for two objects and display the name of the journal having greater price.** ***(Note: Any other logic shall be considered).***	**8M**

	Ans.	#include<iostream.h>
		#include<conio.h>
		class Journal
		{
		char journal_nm[20];
		int ISSN_No;
		float price;
		public:
		void accept();
		void display(Journal);
		};
		void Journal::accept()
		{
		cout<<"\n Enter journal's data:";
		cout<<"\n Name:";
		cin>>journal_nm;
		cout<<"\n ISSN No:";
		cin>>ISSN_No;
		cout<<"\n Price:";
		cin>>price;
		}
		void Journal::display(Journal j2)
		{
		if(price>j2.price)
		{
		cout<<"\n Data of journal having greater price is:";
		cout<<"\n Name:"<<journal_nm;
		cout<<"\nISSN No="<<ISSN_No;
		cout<<"\n Price="<<price;
		}
		else
		{
		cout<<"\n Data of journal having greater price is:";
			Correct definition of class Journal- 2M
			
			Accept and display function with comparison 4M
			
			Correct definition of main() 2M

		cout<<"\n Name:"<<j2.journal_nm;
		cout<<"\n ISSN No="<<j2.ISSN_No;
		cout<<"\n Price="<<j2.price;
		}
		}
		void main()
		{
		Journal j1,j2;
		clrscr();
		j1.accept();
		j2.accept();
		j1.display(j2);
		getch();
		}	

	(b)	**Write a program in C++ to implement single inheritance from following figure:**  ***(Note: Any other logic shall be considered).***	**8M**

	Ans.	#include<iostream.h>
		#include<conio.h>
		class product
		{
		protected:
		int prodid;
		char prodname[20];
		};
		class edible:public product
		{
			Correct definition of class product 3M

		char flavour[20];
		public:
		void accept()
		{
		cout<<"\n Enter product's data:";
		cout<<"\n Id:";
		cin>>prodid;
		cout<<"\n Product name:";
		cin>>prodname;
		cout<<"\n Flavour:";
		cin>>flavour;
		}
		void display()
		{
		cout<<"\n Product's data is:";
		cout<<"\n Id:"<<prodid;
		cout<<"\n Name:"<<prodname;
		cout<<"\n Flavour:"<<flavour;
		}
		};
		void main()
		{
		edible e[5];
		int i;
		clrscr();
		for(i=0;i<5;i++)
		{
		e[i].accept();
		}
		for(i=0;i<5;i++)
		{
		e[i].display();
		}
		getch();
		}
			Correct definition of class edible 3M
			
			Correct definition of main() 2M

	(c)	**Write a program in C++ to accept a string from a user and display its reverse using pointer.** ***(Note: Any other logic shall be considered)***	**8M**

WINTER – 2019 EXAMINATION
MODEL ANSWER

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

	Ans.	#include<iostream.h>
		#include<conio.h>
		#include<string.h>
		void main()
		{
		char str[20],*ptr;
		int l;
		clrscr();
		cout<<"\n Enter a string : ";
		cin>>str;
		l=strlen(str);
		ptr=str;
		while(*ptr!='\0')
		{
		ptr++;
		}
		cout<<"\n Reverse string:";
		while(l!=0)
		{
		ptr--;
		cout<<*ptr;
		l--;
		}
		getch();
		}
			Declaration of string and pointer 1M
			
			Acceptance of string 1M
			
			Calculation of reverse string 3M
			
			Display of reverse string 3M