MODEL ANSWER

WINTER - 2017 EXAMINATION

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)**	**Attempt any SIX of the following:**	**6x2=12**
	(a)	**What is use of scope resolution operator?**	**2M**

	Ans.	It is used to uncover a hidden variable. Scope resolution operator allows access to the global version of a variable. The scope resolution operator is used to refer variable of class anywhere in program.
		:: Variable_name
		OR
		Scope resolution operator is also used in classes to identify the class to which a member function belongs. Scope resolution variable is used to define function outside of class.
		Return_type class_name **::** function_name( )
		{
		}
			Correct use-2M

	(b)	**What is pointer? Give example.**	**2M**
	Ans.	Pointer is a variable that holds memory address of another variable of similar data type. *Example:*	***Pointer definition 1M***

		int a, *ptr; ptr=&a; ptr is a integer data type that holds address of another variable of similar type i.e. a.	***Any example 1M***

	(c) Ans.	**How user can declare member function outside the class?**
		The user can declare member function outside of the class with the help scope resolution operator.
		
		return_type class_name **::** function_name(argument(s))
		{
		Function body;
		}
			2M
			Correct declaration 2M

	(d) Ans.	**Enlist types of constructor.** **Types of constructor:** • Default constructor • Parameterized constructor • Multiple constructor/constructor overloading • Constructor with default argument • Copy constructor	**2M** ***Any four types ½M each***
	(e) Ans.	**What is static polymorphism?** In overloaded functions, when an appropriate member function is selected by matching arguments at compile time then it is known as static polymorphism. **OR** Linking of function call with its definition at the compile time is called as static polymorphism.	**2M** ***Correct definition 2M***
	(f) Ans.	**State different types of visibility mode in inheritance.** There are three visibility modes in inheritance: 1. Private 2. Public 3. Protected	**2M** ***Correct list 2M***

	(g) Ans.	**Give syntax of create pointer to object.**
		Syntax:
		class_name *pointer_variable, object_name;
		pointer_variable=&object_name;
		OR
		class_name object_name;
		class_name *pointer_variable=& object_name;
			2M
			Correct syntax 2M

MODEL ANSWER

WINTER - 2017 EXAMINATION

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

	(h) Ans.	**What do you mean by overloaded constructor?** When more than one constructor function is defined in a class, it is referred as overloaded constructor. Each overloaded constructor function is invoked with respect to arguments pass in the function call.	**2M** ***Correct definition 2M***

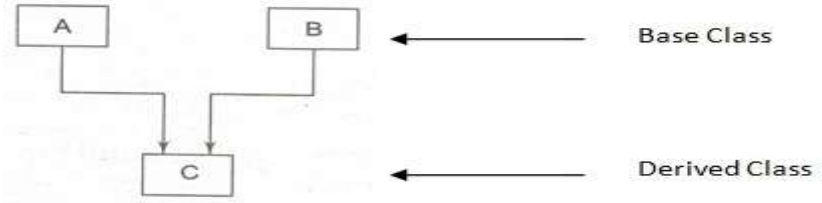
1.	**(B) (a) Ans.**	**Attempt any TWO of the following:**
		What is parametrized constructor? Explain with example.
		A constructor that can take arguments is known as parameterized constructor.
		In some applications, it may be necessary to initialize the various data members of different objects with different values when they are created. So parameterized constructor is used to achieve this by passing arguments to the constructor function when the objects are created.
		
		Example:
		class ABC
		{
		int m;
		public:
		ABC(int x)
		{
		m=x;
		}
		void put()
		{
		cout<<m;
		}
		};
		
		void main()
		{
		ABC obj(10);
		obj.put();
		}
		In the above example, constructor 'ABC (int x)' is a parameterized constructor function that accepts one parameter. When 'obj' object is created for class 'ABC', parameterized constructor will be invoked and data member 'm' will initialize with the value 10 which is passed
			2x4=8
			4M
			Explanation 2M
			Example 2M

as an argument when object is created. Member function 'put' displays the value of data member 'm'.

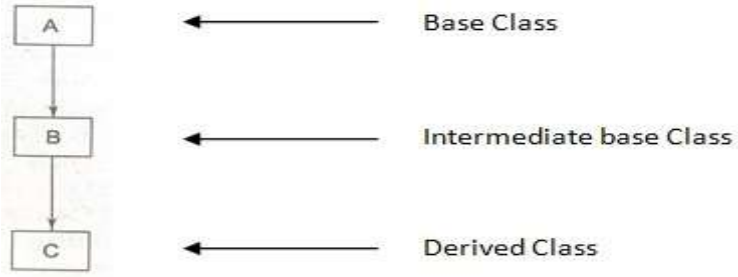
(b) **What is multiple inheritance? What is multilevel inheritance? What is difference between them?** — **4M**

Ans.

Multiple Inheritance: When A derived class is derived from more than one base class then it is known as multiple inheritance. A derived class can inherit the attributes of all base classes from which it is derived. — ***Multiple inheritance 1M***



Multi-level Inheritance: - When a derived class is derive from a intermediate base class which in turn derived from another base class then this type of inheritance is known as multi level inheritance. The derived class can inherit properties of base classes from each level. — ***Multi-level inheritance 1M***



Difference:

Multiple Inheritance	Multi-Level Inheritance
In this there exists more than one base class to derived class.	In this type of inheritance there exists only one base class to derived class.
It does not have intermediate class.	It has intermediate class.
It can go up to only one level.	It can go up to any level based on programming need.
The derived class inherits some property of all base classes.	In this, last derived class will inherit only public property of

Any two relevant Differences 1M each

		all class including intermediate class.	

	(c)
	Ans.
		What is destructor? How destructor is declared? When destructor is invoked?
		Destructor: A destructor is a special member function used to destroy the objects that are created by a constructor.
		
		Declaration:
		Destructor is declared in public section of a class. It is member function whose name is same as the class name but proceeded by a tilde (~)
		~destructor_name( )
		{
		Body of function
		}
		
		Invokes:
		It will be invoked implicitly by the compiler upon exit from the program (or block or function) to clean up storage that is no longer accessible.
			4M
			Definition of destructor 2M
			Declaration 1M
			Invoke 1M

2.	**(a)** **Ans.**	**Attempt any FOUR of the following:** **What are the differences between structure and class?** ***(Note: Any other relevant point shall be considered)***	**4x4=16** **4M** ***Any four differences 1M each***

Structure	Class
1. Structure contains logically related data items which can be of similar type or different type.	1. Class is a way of binding data and functions together in one single unit. It is a collection of data members and member functions.
2. In structure data is not hidden from external use.	2. It allows data and functions to be hidden from external use.
3. In Structure all members by default are public.	3. In Class all members are by default are private.
4. In Structure structure_variable is created.	4. In class object is created.
5. **Syntax:** struct structure_name {	**5. Syntax:** class class_name {

		datatype variable1; datatype variable2; .. .. }structure_variable;	access specifier: declare data members; declare member functions; };

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

	(b)
	Ans.
		What is pure virtual functions? Write down the rules used for pure virtual function.
		(Note: Any other relevant rules shall be considered)
		A **pure virtual function** is a function which is declared in a base class and which does not have definition relative to the base class. In such cases, the compiler requires each derived class to either defined in a derived class or is re-declared as pure virtual function. A pure virtual function in base class only serves as a placeholder. Such functions are also called as 'do-nothing' functions.
		
		Ex:-
		class ABC
		{
		public:
		virtual void display( )=0;
		};
		
		Rules:-
		1. A pure virtual function in base class does not have definition relative to base class.
		2. A class containing pure virtual functions cannot be used to declare any objects of its own.
			4M
			
			Description of pure virtual function 2M
			
			Any two rules 1M each

	(c) **Ans.**	**What is base class? What is derived class? Give example.**	**4M**

		Base class: In inheritance a new class is derived from an old class. The old/existing class is referred as base class whose properties can be inherited by its derived class.
		
		Derived class: In inheritance a new class is derived from an old class. The new class is referred as derived class. The derived class can inherit all or some properties of its base class. With derived class object, one can access members of itself as well as base class.
		
		Example:
		class base
		{
		public:
		void putb()
		{
		cout<<"base class";
		}
		};
		
		class derived:public base
		{
		public:
		void putd()
		{
		cout<<"derived class";
		}
		};
		void main( )
		{
		derived d;
		d.putb();
		d.putd();
		}
			Definition of base class 1M
			Definition of derived class 1M
			
			Any example 2M

	(d) Ans.	**Write down characteristics of object oriented programming.** **Characteristics of object oriented programming are:** • Emphasis is on data rather than procedure. • Programs are divided into objects. • Data structures are designed such that they characterize the objects.	**4M** ***Any four characteristics 1M each***

		• Functions that operate on the data of an object are tied together in the data structure. • Data is hidden & cannot be accessed by external functions. • Objects may communicate with each other through functions. • New data and functions can be easily added whenever necessary. • Follows bottom-up approach in program designing.	

	(e) Ans.	**Explain the need of static member function with example.**
		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;
		void main()
		{
		test t1,t2;
		t1.setcount();
			4M
			
			Explanation 2M
			
			Any example 2M

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

	(f) Ans.	**What is this pointer? Explain with example.**
		'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<iostream.h>
		class sample
		{
		int a;
		public:
		void setdata(int x)
		{
		this ->a=x;
		}
		void putdata()
		{
		cout<<this ->a;
		}
		void show()
		{
		this->putdata();
		}
		};
		void main()
		{
		sample s;
		s.setdata(100);
			4M
			
			Explanation 2M
			
			Any Example 2M

		s.show();
		}
		In the above example, this pointer is used to represent object s when setdata() and show() functions are called.	

3.	**(a)** **Ans.**	**Attempt any FOUR of the following:** **Explain different operator used in C++.** **:: Scope resolution operator:** This operator allows access to the global version of a variable. Scope resolution operator is also used in classes to identify the class to which a member function belongs. **::* -Pointer-to-member declarator:** This is used to declare pointer to a member of a class. **->* or .* -Pointer-to-member operator**: This is used to access a member using a pointer to the object and pointer to that member. **Delete** - Memory release operator: An object can be destroyed by using delete operator. **Endl** - Line feed operator: It causes a linefeed to be inserted. **New** -Memory allocation operator: An object can be created by using new. It is used to allocate memory to the object. **<< Insertion or put to operator:** It inserts contents of the variable on its right to the object on its left. It is used with cout statement to display output on screen. **>> Extraction or get from operator**: It extracts the value from the keyboard and assigns it to the variable on its right. It is used with cin statement to input data.	**4x4=16** **4M** ***Any 4 operators 1M each***
	(b) **Ans.**	**Write a program which implement the concept of overloaded constructor.** #include<iostream.h> #include<conio.h> class integer { int m,n; public:	**4M** ***Correct logic 2M***

		integer()
		{
		m=0;
		n=0;
		} //default constructor 1
		integer(int a, int b)
		{
		m=a;
		n=b;
		} //Parameterized constructor 2
		integer(integer &i)
		{
		m=i.m;
		n=i.n;
		} //copy constructor 3
		};
		void display()
		{
		cout<<a<<endl;
		cout<<b;
		}
		void main()
		{
		integer i1,i2(30,40);
		i1.display();
		i2.display();
		integer i3(i2);
		i3.display();
		getch();
		}
			Correct syntax 2M

	(c) Ans.	**What is virtual base class? Explain with example.** **Virtual base class:** Consider a situation where all three kinds of inheritance, namely, multilevel, multiple, hierarchical inheritance, are involved. This illustrated in fig a. the child has two direct base classes, "parent1" & "parent2" which themselves have a common base class "grandparent". The child inherits the traits of "grandparent" via two separate paths. It can also inherit directly as shown by broken line. The "grandparent" sometimes referred to as indirect base class.	**4M** ***Explanation 2M***

All the public & protected members of "grandparent" are inherited into "child" twice, first via "parent1" & again via "parent 2". This means, "child" would have duplicate sets of the members inherited from "grandparent. This introduces ambiguity & should be avoided. The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

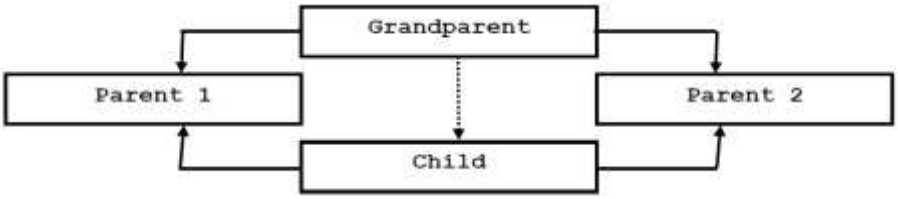


Fig. a: Virtual Base Class

Example:

```
#include<iostream.h>
#include<conio.h>
class student
{
int rno;
public:
void getnumber()
{
cout<<"Enter Roll No:";
cin>>rno;
}
void putnumber()
{
cout<<"\n\n\t Roll No:"<<rno<<"\n";
}
};
class test: virtual public student
{
public:
int part1,part2;
void getmarks()
{
```

Any Example 2M

```
cout<<"Enter Marks\n";
cout<<"Part1:";
cin>>part1; cout<<"Part2:";
cin>>part2;
}
void putmarks()
{
cout<<"\t Marks Obtained\n";
cout<<"\n\t Part1:"<<part1;
cout<<"\n\tPart2:"<<part2;
}
};
class sports: public virtual student
{
public:
int score;
void getscore()
{
cout<<"Enter Sports Score:";
cin>>score;
}
void putscore()
{
cout<<"\n\t Sports Score is:"<<score;
}
};
class result: public test, public sports
{
int total;
public:
void display()
{
total=part1+part2+score;
putnumber();
putmarks();
putscore();
cout<<"\n\t Total Score:"<<total;
}
};
```

		void main()
		{
		result obj;
		clrscr();
		obj.getnumber();
		obj.getmarks();
		obj.getscore();
		obj.display();
		getch();
		}	

	(d)
	
	Ans.
		Write a program which overload unary++ operator by using friend function.
		(Note: Any relevant program shall be considered)
		#include<iostream.h>
		class space{
		int x;
		int y;
		int z;
		public:
		void getdata(int a,int b,int c);
		void display(void);
		friend void operator++(space &s);
		};
		void space :: getdata(int a,int b,int c){
		x=a;
		y=b;
		z=c;
		}
		void space :: display(void){
		cout<<x<<" ";
		cout<<y<<" ";
		cout<<z<<"\n";
		}
		void operator++(space &s){
		s.x++;
		s.y++;
		s.z++;
		}
		int main(){
			4M
			
			Correct logic 2M
			
			Correct syntax 2M

		space s;
		s.getdata(10,-20,30);
		cout<<"S : ";
		s.display();
		++s;
		cout<<"S :";
		s.display();
		return 0;
		}	

	(e) Ans.	**Explain the derived class access by pointer.**
		When base class and its derived class both contains same function name then the function in base class is declared as virtual using keyword virtual preceding its normal declaration. When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of the pointer. Thus by making the base pointer to point to different objects we can execute different versions of the virtual function.
		
		Example:
		#include<iostream.h>
		class Base
		{
		public:
		virtual void show( )
		{
		cout<<"\n show base";
		}
		};
		class derived : public Base
		{
		public:
		void show( )
		{
		cout<<"\n show derived";
		}
		};
		void main( )
		{
			4M
			
			Relevant explanation 2M
			
			Any example 2M

		Base B;
		derived D;
		Base *bptr;
		bptr=&B;
		bptr->show( );
		bptr=&D;
		bptr->show( );
		}	

	(f) Ans.	**Explain how to pass object as function argument.**
		An object may be used as function arguments in two methods:-
		i) A copy of the entire object is passed to the function.
		ii) Only the address of the object is transferred to the function.
		
		1) **Pass-by-value**
		Since a copy of the object is passed to the function, any changes made to the object inside the function do not affect the object used to call the function.
		
		2) **Pass-by-reference**
		When an address of the object is passed the called function works directly on the actual object used in the call. This means that any changes made to the object inside the function will reflect in the actual object.
		
		Example:
		Following program illustrates the use of object as function arguments. It performs the addition of time in the hour & minute format.
		# include<iostream.h>
		class time
		{
		int hours;
		int minutes;
		public:
		void gettime(int h, int m)
		{
		hours = h;
		minutes = m;
		}
		void puttime(void)
			4M
			
			Explanation 2M
			
			Example 2M

		{
		cout<< hours << "hours and: ";
		cout<< minutes << " minutes " << "\n";
		}
		void sum (time, time);
		};
		void time :: sum (time t1, time t2)
		{
		minutes =t1.minutes + t2.minutes ;
		hours = minutes / 60;
		minutes = minutes%60;
		hours = hours = t1.hours + t2.hours;
		}
		main()
		{
		time T1, T2, T3;
		T1.gettime(2, 30);
		T2.gettime(3, 45);
		T3.sum(T1, T2);
		cout<< " T1 = ";
		T1.puttime();
		cout<< " T2 = ";
		T2.puttime();
		cout<< " T3= ";
		T3.puttime();
		}
		An object can also be passed as argument to a non-member function but, such functions can have access to the public member function only through the object passed as arguments to it.	

4.	**(a) Ans.**	**Attempt any FOUR of the following:** **Explain hybrid inheritance with example.** **Description:** "Hybrid Inheritance" is a method where one or more types of inheritance are combined together and used. Hybrid Inheritance is combination of Single, Multiple, Hierarchical and Mutilevel Inheritance. We can use any combination to form hybrid inheritance only single level inheritance cannot be combined with multi-level inheritance as it will result into multilevel inheritance.	**4x4=16** **4M** ***Description 2M***

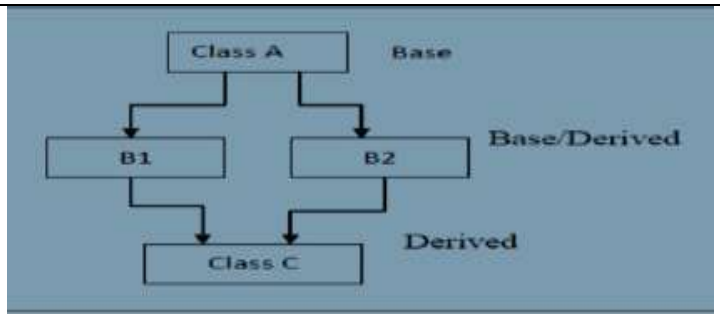
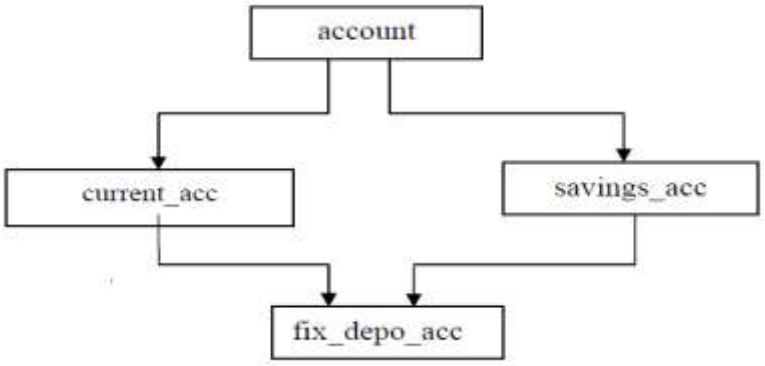


Fig: Hybrid Inheritance

Above figure shows graphical representation of hybrid inheritance

Example:



Above diagram is an example of hybrid inheritance. Class current_acc and class savings_acc derive common members such as name and acc_no from class account. Then all the data of account is combined in class fix_depo_acc.

Any relevant example /program 2M

OR

Program as an example:

```
#include<iostream.h>
#include<conio.h>
class account
{
protected:
char acc_name[10];
long ph_no;
};
```

```
class savings_acc:virtual public account
{
protected:
int s_acc_no;
};
class current_acc:virtual public account
{
protected:
int c_acc_no;
};
class fix_depo_acc:public savings_acc,current_acc
{
int fix_depo_acc_no;
public:
void getdata()
{
cout<<"\n Enter account holder name : ";
cin>>acc_name;
cout<<"\n Enter phone number : ";
cin>>ph_no;
cout<<"\n Enter savings account number : ";
cin>>s_acc_no;
cout<<"\n Enter current account number : ";
cin>>c_acc_no;
cout<<"\n Enter fixed deposit account number : ";
cin>>fix_depo_acc_no;
}
void putdata()
{
cout<<"\n\n Account holder name : "<<acc_name;
cout<<"\n phone number : "<<ph_no;
cout<<"\n Savings account number : "<<s_acc_no;
cout<<"\n Current account number : "<<c_acc_no;
cout<<"\n Fixed deposit account number : "<<fix_depo_acc_no;
}
};
void main()
{
fix_depo_acc f;
```

		clrscr();
		f.getdata();
		f.putdata();
		getch();
		}	

	(b) Ans.	**What is copy constructor? Explain with example.**
		Copy constructor:
		The **copy constructor** is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to:
		• Initialize one object from another of the same type.
		• Copy an object to pass it as an argument to a function.
		• Copy an object to return it from a function.
		If a copy constructor is not defined in a class, the compiler itself defines one. If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor.
		
		Example:
		#include<iostream.h>
		class point
		{
		private:
		int x, y;
		public:
		point(int x1, int y1)
		{
		x = x1;
		y = y1;
		}
		// copy constructor
		point(point &p2)
		{
		x = p2.x;
		y = p2.y;
		}
		int getX()
		{
		return x;
			4M
			
			Explanation 2M
			
			Any example 2M

```
        }
          int getY()
          {
          return y;
        }
};
int main()
{
          point p1(10, 15); // Normal constructor is called here
          point p2 = p1; // Copy constructor is called here
          // Let us access values assigned by constructors
          cout<< "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
          cout<< "\np2.x = " << p2.getX() << ", p2.y = " <<
p2.getY();
          return 0;
}
```

(c) Ans. **Explain the structure of C++ program.** **4M**

General C++ program has following structure.

INCLUDE HEADER FILES
DECLARE CLASS
DEFINE MEMBER FUNCTIONS
DEFINE MAIN FUNCTION

Structure 1M

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.

Explanation 3M

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

		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.	**What is friend function? Explain 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 of the class in which it is declared. It is called without any object. The class members are accessed with the object name and do membership operator inside the friend function. It accepts objects as arguments.
		
		Syntax for declaring friend function:-
		friend return_type function_type(parameter1,parameter2,…, parameter n);
		
		Syntax for calling friend function: -
		function_name(parameter1,parameter2,,parameter n);
		
		Example:
		#include<iostream.h>
		#include<conio.h>
		class base {
		int val1, val2;
		public:
		void get()
		{
		cout<< "Enter two values:";
		cin>> val1>>val2;
		}
		friend float mean(base ob);
		};
		float mean(base ob)
		{
			4M
			
			Explanation 2M
			
			Any Example 2M

		return float(ob.val1 + ob.val2) / 2;
		}
		void main()
		{
		clrscr();
		base obj;
		obj.get();
		cout<< "\n Mean value is : " << mean(obj);
		getch();
		}	

	(e)
	Ans.
		What is inheritance? What is use of inheritance? Explain.
		Inheritance:
		What is inheritance:
		The mechanism of deriving new class from an old/existing one is called inheritance.
		OR
		Inheritance is the process by which objects of one class acquired the properties of objects of another classes.
		
		Syntax:
		class derived-class-name: visibility-mode base-class-name
		{
		------//
		-----// members of derived class
		-----//
		};
		
		Use:
		Reusing the properties of the existing class.
		The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as the base class and the new one is called as the derived class or subclass. Derive class can inherit all or some properties of its base classes.
			4M
			
			Explanation 2M
			
			Use 2M

	(f) **Ans.**	**Write a program which implement the concept of pointer to array.** ***(Note: Any relevant program shall be considered)*** #include<iostream.h> #include<conio.h> void main()	**4M** ***Correct logic 2M***

		{
		int x[10],*xptr,i;
		clrscr();
		cout<<"\nEnter 10 Numbers";
		for(i=0;i<10;i++)
		{
		cin>>x[i];
		}
		xptr=&x[0];
		while(*xptr!='\0')
		{
		cout<<*xptr<<" ";
		xptr++;
		}
		getch();
		}
			Correct syntax 2M

5.	**(a) Ans.**	**Attempt any FOUR of the following:** **Write down rules used for operator overloading.** **Rules for overloading operators:** 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 data type. 3. We can't change the basic meaning of an operator. That is to say, we can't redefine the plus(+) operator to subtract one value from other. 4. Overloaded operators follow the syntax rules of the original operators. They can't be overridden. 5. There are some operators that can't be overloaded. 6. We can't use friend functions to overload certain operators. However, member function scan be used to overload them. 7. Unary operators overloaded by means of member function take no explicit arguments and return no explicit values, but, those overloaded by means of the friend function, take one reference argument (the object of the relevant class). 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	**4x4=16** **4M** ***Any four rules 1M each***

		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.	

	(b) Ans.	**What is structure? How user declare structure? Give example.**
		Structure is a collection of different data types written under a common name. It is a user defined data type.
		To Define a structure following Syntax is use:-
		struct structure_name
		{
		data_type variable 1;
		data_type variable 2;
		data_type variable n;
		};
		
		Example:
		#include <iostream.h>
		struct Person
		{
		char name[50];
		int age;
		float salary;
		
		};
		
		void main()
		{
		Person p1;
		cout<< "Enter Full name: ";
		cin>>p1.name;
		cout<< "Enter age: ";
		cin>> p1.age;
		cout<< "Enter salary: ";
		cin>> p1.salary;
		cout<< "\n Displaying Information." <<endl;
		cout<< "Name: " << p1.name <<endl;
		cout<<"Age: " << p1.age <<endl;
		cout<< "Salary: " << p1.salary;
			4M
			
			Structure Definition 1M Syntax 1M
			
			Example 2M

		}
		How to define virtual function? Give example.
		A virtual function is a member function that is declared within a base class and redefined by a derived class.
		
		To define virtual function following syntax is used:
		Class baseclass
		{
		virtual function_name()
		{
		}
		};
		Class deriveclass
		{
		function_name()
		{
		}
		};
		
		Example:
		#include<iostream.h>
		class Base
		{
		public:
		virtual void show( )
		{
		cout<<"\n show base";
		}
		};
		class Derived : public Base{
		public:
		void show( )
		{
		cout<<"\nshow derived";
		}
		};
		void main( )
		{
		Base B;
			4M
			
			Syntax 1M
			
			Example 2M

	(c) Ans.		

Derived D;
Base *bptr;
bptr=&B
bptr→show( );
bptr=&D;
bptr→show( );
}

(d) **What is difference between procedure oriented programming and object oriented programming.** **4M**

Ans.

Sr. No.	PROCEDURE ORIENTED PROGRAMMING (POP)	OBJECT ORIENTED PROGRAMMING (OOP)
1	Focus is on doing things (procedure).	Focus is on data rather than procedure.
2	Large programs are divided into multiple functions.	Programs are divided into multiple objects.
3	Data move openly around the system from function to function.	Data is hidden and cannot be accessed by external functions.
4	Functions transform data from one form to another by calling each other.	Objects communicate with each other through function.
5	Employs top-down approach in program design.	Employs bottom-up approach in program design
6	Procedure oriented approach is used in C language.	Object oriented approach is used in C++ language.

Any four differences 1M each

(e) **Write a program which perform arithmetic operation using pointer.** **4M**

(Note: Any relevant program shall be considered)

Ans.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int num[5]={56,75,22,18,90};
int ptr;
```

Correct program 4M

		int i;
		cout<<"array elements are::";
		for(i=0;i<5;i++)
		ptr=num;
		cout<<"value of ptr::"<<*ptr;
		cout<<"\n";
		ptr++;
		cout<<"value of ptr++::"<<*ptr;
		cout<<"\n";
		ptr--;
		cout<<"value of ptr--::"<<*ptr;
		cout<<"\n";
		ptr=ptr+2;
		cout<<"value of ptr+2::"<<*ptr;
		cout<<"\n";
		ptr=ptr-1;
		cout<<"value of ptr-1::"<<*ptr;
		cout<<"\n";
		ptr+=3;
		cout<<"value of ptr+=3::"<<*ptr;
		cout<<"\n";
		getch();
		}	

	(f) **Ans.**	**What is polymorphism? Enlist different types of polymorphism. What are the differences between them?** **Polymorphism:** It is a feature of object oriented programming which allows a programmer to have a more than one function having same name but different /same parameters but performs different/distinct operation. Different Types of Polymorphism: • Runtime Polymorphism o Function overloading o Operator Overloading • Compile Time polymorphism o Virtual Function	**4M** ***Definition 1M*** ***List 1M***

Difference between Types of Polymorphism:

Runtime Polymorphism	Compile Time polymorphism
In this polymorphism, selection of appropriate function is done at run time.	In this polymorphism, an object is bound to its function call at compile time
Function to be called is unknown until appropriate selection is made	Functions to be called are known well before.
This requires use of pointers to object	This does not require use of pointers to objects
Function calls execution are slower	Function calls execution are faster
Also called as late binding, dynamic binding.	Also called as early binding, Static binding
It is implemented with virtual function.	It is implemented with operator overloading or function overloading

Any 2 Points of comparison 1M each

6. **Attempt any TWO of the following:** **2x8=16**

(a) **What is function? What is call by value? What is call by reference? What is the difference between them?** **8M**

Ans.

Function:

A **function** is a group/set of statements/instruction that performs a specific task. It can be system defined i.e. clrscr(), getch(), or user defined i.e. add(), sub().

Function Definition 1M

Call by Value:

In call by value, original value is not modified.

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Call by Value 2M

Call by Reference:

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Call by Reference 2M

Difference between Call by Value and Call by Reference:

Call by Value	Call by Reference
Call by value method of calling function pass value of parameters as arguments.	Call by reference method of calling function pass address of parameters as arguments.
Call by value method copy of actual parameter is created	Call by value method no copy of actual parameter is created, address of actual parameters is passed
Processing inside function does not affect actual parameters. Function works only on copy of parameters	Processing inside the function affects actual parameters as operations are done only on actual parameters.

Example: swap(a,b); //function call
void swap(inta,int b)// function definition
{
 	
}
	Example: swap(&a,&b); //function call
	void swap(int *a,int *b)// function definition
	{
	}

Any 3 points 1M each

(b) **What is constructor? How user can declared constructor in derived class? Explain with example.** **8M**

Ans.

Constructor:-

Constructor is a special member function which has same name as a class name and is used to initialize object during compile time of program.

Definition 2M

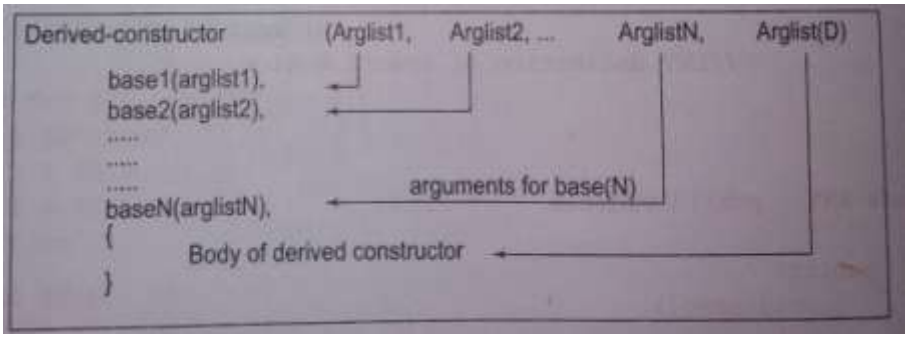
Declaring constructor in derived class

If a base class contains a constructor with one or more arguments then it is mandatory for the derived class to have a constructor and pass the argument to the base class constructor.

A header line of derived constructor function contains two parts

Declaration 2M

separated by a colon (:)
First part provides the declaration of the arguments that are passed to the derived constructor and the second part lists the function calls to the base constructors.



Example:

```
#include<iostream.h>
#include<conio.h>
class base
{
int a;
public:
base(int x)
{
a=x;
}
void displaybase()
{
cout<<a;
}
};
class derived:public base
{
int b;
public:
derived (int x,int y):base(x)
{
b=y
}
```

Example 4M

```
void display()
{
cout<<b;
}
};
void main()
{
derived d(2,5);
d.displaybase();
d.display();
getch();
}
```

(c) **Write a program which concate and reverse string by using pointer to string.** **8M**

Ans.

```
#include<iostream.h>
#include<conio.h>
void main()
 {
char str1[20], str2[20], rev[20],*p1,*p2;
int cnt=0;
clrscr();
cout<<"\n Concatenation of String";
cout<<"\n Enter two Strings";
cin>>str1>>str2;
   p1=&str1[0];
   p2=&str2[0];
while(*p1!='\0')
    {
        p1++;
    }
while(*p2!='\0')
    {
        *p1=*p2;
        p1++;
        p2++;
    }
  *p1='\0';
cout<<"\nConcatenated String is "<<str1;
cout<<"\nReverse of String";
```

Concatenation 4M

Reverse String 4M

```
   p2=&str2[0];
while(*p2!='\0')
     {
          p2++;
          cnt++;
     }
p2--;
   p1=&rev[0];
while(cnt>0)
     {
          *p1=*p2;
          p2--;
          p1++;
     }
   *p1='\0';
cout<<"\n Reverse String is :\t"<<rev;
getch();
  }
```