QNO	SCORING INDICATOR	SPLIT UP SCORE	SUB TOTAL	TOTAL
	PART A			
I(1)	Char string[4]="ABC"	2	2	10
I(2)	Expression is a combination of operators, constants and variables arranged as per the rules of the language. Example A+B	2	2	
I(3)	A constructor is a 'special' member function whose task is to initialize the objects of its class. Its name is the same as the class name.	2	2	
I(4)	Class member access operators(. , .*). scope resolution operator (::). Size operator (slzeof). Conditional operator (?:).	2	2	
I(5)	Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. Exceptions allow a method to react to exceptional circumstances and errors (like runtime errors) within programs by transferring control to special functions called handlers	2	2	
	PART B			
II(1)	a. Emphasis is on data rather than procedure. b. Programs are divided into what are known as objects. c. Data structures are designed such that they characterize the objects. d. Functions that operate on the data of an object are tied together in the data st.rue· e. Data is hidden and cannot be accessed by external functions. f. Objects may communicate with each other through functions. g. New data and functions can easily added whenever necessary. h. Follows bottom-up approach in program design.	6	6	30
II(2)(a)	**Do-While** The do-while is an exit-controlled loop. Based on a condition, the control is transferred back to a particular point in the program. The syntax is as follows: do {	3		

	action 1;
	}
	while(condition is true);
	act1on2;			

II(2)(b)	While
	This is also a loop structure, but is an entry-controlled one. The syntax is as follows:
	while{condition is true)
	{
	action1;
	}
	action2;
		3	6	6

II(3)	Conversion of an expression of a given type into another type is called as type casting. Type Casting is a mechanism which enables a variable of one datatype to be converted to another datatype. When a variable is typecast into a different type, the compiler basically treats the variable as of the new data type. C++ permits explicit type conversion or variables or expressions using the type cast operator. Syntax is (type) expression There are other casting operators supported by C++, they are listed below – • **const_cast<type> (expr)** – The const_cast operator is used to explicitly override const and/or volatile in a cast. The target type must be the same as the source type except for the lteration of its const or volatile attributes. This type of casting manipulates the const attribute of the passed object, either to be set or removed. • **dynamic_cast<type> (expr)** – The dynamic_cast performs a runtime cast that verifies the validity of the cast. If the cast cannot be made, the cast fails and the expression evaluates to null. A dynamic_cast performs casts on polymorphic types and can cast a A* pointer into a B* pointer only if the object being pointed to actually is a B object. • **reinterpret_cast<type> (expr)** – The reinterpret_cast operator changes a pointer to any other type of pointer. It also allows casting from pointer to an integer type and vice versa.	6	6	

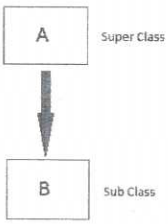
	• **static_cast<type> (expr)** – The static_cast operator performs a nonpolymorphic cast. For example, it can be used to cast a base class pointer into a derived class pointer.			

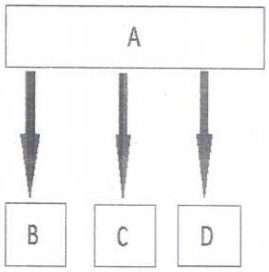
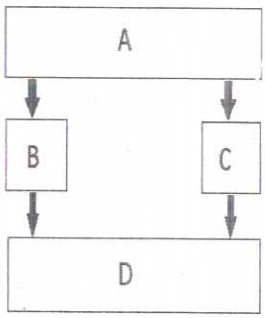
II(4)	C++ allows to call a function without specifying all its arguments. In such cases, the function assigns a default value to the parameter which does not have a matching argument in the function call. Default values are specified when the function is declared. The compiler looks at the prototype to see how many arguments a function uses and alerts the program for possible default values. A default argument is checked for type at the time of declaration and evaluated at the time of call. Key point is that only the trailing arguments can have default values and therefore we must add defaults from right to left.
	#include<iostream.h>
	`#include<conio.h>`
	`int Add(int x, int y=20, int z=30)`
	`{`
	`return x + y + z;`
	`}`
	`void main()`
	{int rs;
	`rs = Add(5);`
	`cout<<"\n\tThe sum is : "<<rs;`
	`rs = Add(4,8);`
	`cout<<"\n\tThe sum is : "<<rs;`
	`rs = Add(7,3,4);`
	`cout<<"\n\tThe sum is : "<<rs;`
	}
		6	6	6

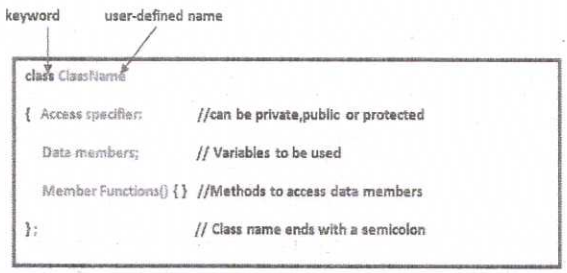
II(5)	C++ function call by reference. The call by reference method of passing arguments to a function copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call.
	#include <iostream>
	using namespace std;
	void swap(int& x, int& y)
	{ int z = x;
	x = y;
	y = z;
		6	6	6

	}
	int main()
	{ int a = 45, b = 35;
	cout << "Before Swap\n";
	
	cout << "a = " << a << " b = " << b << "\n";
	swap(a, b);
	cout << "After Swap with pass by reference\n";
	cout << "a = " << a << " b = " << b << "\n";
	}			

II(6)	#include <iostream>
	using namespace std;
	class Check{ private: int i; public: Check(): i(0) { }
	void operator ++() { ++i; }
	void Display() { cout << "i=" << i << endl; }};
	int main()
	{ { Check obj; // Displays the value of data member i for object obj
	obj.Display(); // Invokes operator function void operator ++( )
	++obj; // Displays the value of data member i for object obj
	obj.Display(); return 0;}			

II(7)	Single Inheritance  Single Inheritance In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance. Hierarchical Inheritance in C++ In this type of inheritance, multiple derived classes inherits from a single base class			

	 Hybrid Inheritance in C++ Hybrid Inheritance is combination of two or more inheritances . i e single, multiple,multilevel or hierarchical Inheritances. Figure shows is combination of Hierarchical and Mutilevel Inheritance. 			
	PART C			
III(a)(1)	1. Identifiers Identifiers refer to the names of variables, functions. arrays, classes, etc. created by the programmer. The following are rules for naming identifiers in C++: 2. The name cannot start with a digit. 3. Uppercase and lowercase letters are distinct. 4 Declared keyword cannot be used as a variable name.	3	9	60
III(a)(2)	Relational operators ' == ' Checks if the values of two operands are equal or not, if yes then condition becomes true.	3		

	' !=' Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. ' >'Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. '<' Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. >='Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. '<='Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.			
III(a)(3)	'&&' Logical AND operator. If both the operands are non-zero, then condition becomes true. '||' Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. '!' Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	3		
III(b)(1)	Class : The building block of C++ that leads to Object-Oriented programming is a Class. It holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.  			
	2. Structure Structure provides a method for packing together data of different types. It is a user-defined data type with a template that serves to define its data properties. Once the structure type has been defined, we can create variables of that type using declarations that are			

	similar to the built-in type declarations. Example struct student (char name(20) ; int ro11_number; float total_marks; } ; The keyword struct declares student as a new data type that can hold three fields of different data types·. These fields are known as structure members or elements. The identifier student, which is referred to as structure name or structure tag, can be used to create variables of type student. Example: strut student A			

IV(a)(1)	1. if statement
	If we need to execute a block of statements only when a particular condition is met or not met. This is called decision making, as we are executing a certain code after making a decision in the program logic. For decision making in C++, we have four types of control statements (or control structures).
	The if statement is implemented in two forms:
	Simple if statement
	Form 1
	1f (express1on Is true)
	{
	
	actionl;
	
	}
	act ion2;
	act ion3 ;
	The statements inside if parenthesis (usually referred as if body) gets executed only when the given condition is true. If the condition is false then the statements inside if body are completely ignored.
		3		

IV(a)(2)	2. If else statement If we have a condition and we want to execute a block of code if condition is true and execute another piece of code if the same condition is false. This can be achieved in C++ using if-else statement. if(condition) { Statement(s);	3		

	}
	else {
	Statement(s);
	}
	The statements inside "if" would execute if the condition is true, and the statements inside "else" would execute if the condition is false.			

IV(a)(3)	3. switch
	
	This is a multiple-branching statement where, based on a condition, the control is transferred
	to one of the many possible points. This is implemented as follows:
	switch (n)
	case constant1:
	// code to be executed if n is equal to constant1;
	break;
	case constant2:
	// code to be executed if n is equal to constant2;
	break;
	default:
	// code to be executed if n doesn't match any constant
	}
	When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.
		3	9	

IV(b)	Integer Character Boolean Floating Point Double Floating Point Void Wide Character	6	6	
V(a)	A constructor is a 'special' member function whose task is to initialize the objects of its class. It is special because its name is the 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 values of data members of the class. However, in practice it may be necessary to initialize the various data elements of different objects with different values when they are created. C++ permits us to achieve this objective by passing arguments to the constructor function when the objects are created.	6	6	15

	The constructors that can take argument & are called parametrized constructors.
	class integer
	{
	int m. n;
	public:
	tnteger (int x, int y);//parameterized constructor
	
	};
	integer :: tnteger(int x, int y)
	{
	m=x;
	n=y;}			

V(b)	#include <iostream>
	using namespace std;
	class student { private:
	int rollno;
	char name[20];
	float percentage;
	public: //member function to read details
	void readDetails()
	{ cout << "Enter Roll number:"; cin >> rollno;
	cout << "Enter Name:"; cin >> name; cout << "Enter Percentage:";
	cin >> percentage; }
	//member function to display details
	void printDetails(){ cout << "Student details:\n"; cout <<
	"RollNo=" << rollno; cout << ", Name=" << name; cout << ",
	Percentage=" << percentage; }
	}; int main()
	{ student std[5];//array of students object
	int n, i; cout << "Enter total number of students: "; cin >> n;
	for(i=0; i<n; i++)
	{ cout << "Enter details of student " << i+1 << ":\n";
	std[i].readDetails(); }
	for(i=0; i<n; i++)
	{ cout << "\nDetails of student " << i+1 << ":\n";
	std[i].printDetails();
	} return 0; }
		9	9	

V1 (a)	A copy constructor is a member function which initializes an object using another object of the same class.
	Syntax
	Classname(const classname & objectname)
	{
	}
		9	9	15

	As it is used to create an object, hence it is called a constructor. And, it creates a new object, which is exact copy of the existing copy, hence it is called copy constructor.
	#include <iostream>
	using namespace std;
	class A
	{
	public:
	int x;
	A(int a) // parameterized constructor.
	{
	x=a;
	}
	A(A &i) //copy constructor
	{
	x =i.x;
	}
	};
	int main()
	{
	A a1(20);//Calling the parameterized onstructor.
	Aa2(a1);/Calling the copy constructor.
	cout<<a2.x;
	
	return 0;
	}			

VI(b)	1. Only existing operators can be overloaded 2.The overloaded operators , at least must have one operand that is of user defined type 3.There are some operators cannot be overloaded 4. We cannot use friend function to overload certain operators 5. We cannot change the basic meaning of an operators 6.Overloaded operators follow the syntax rules of original operators	6	6	

VII(a)	#include<iostream.h>
	#include<string.h>
	class comp {
	public:
	char *s;
	void getstring(char *str)
	{
	strcpy(s,str);
		6	6	15

	}
	void friend operator==(comp, comp);
	};
	void operator==(comp ob, comp ob1)
	{
	if(strcmp(ob.s,ob1.s)==0)
	cout<<"\nStrings are Equal";
	else
	cout<<"\nStrings are not Equal";
	}
	void main()
	{
	comp ob, ob1;
	char *string1, *string2;
	clrscr();
	cout<<"Enter First String:";
	cin>>string1;
	ob.getstring(string1);
	cout<<"\nEnter Second String:";
	cin>>string2;
	ob1.getstring(string2);
	//Call Equality Operator
	ob==ob1;
	}			

VII(b)(1)	Multi level
	n C++ programming, not only you can derive a class from the base class but you can also derive a class from the derived class. This form of inheritance is known as multilevel inheritance.
	class A
	{
	};
	class B: public A
	{
	... .
	};
	class C: public B
	{
	... ...
	};
	Here, class *B* is derived from the base class *A* and the class *C* is derived from the derived class *B*.
	#include <iostream>
	using namespace std;
	class A
	{
	public:
		4.5	9	

	void display()
	{
	cout<<"Base class content.";
	}
	};
	class B : public A
	{
	public:
	void display1()
	{
	cout<<"\nClass B content.";
	}
	};
	class C : public B
	{
	public:
	void display2()
	{
	cout<<"Class C content.
	}
	};
	int main()
	{
	C c;
	c.display();
	c.display1();
	return 0;
	}			

VII(b)(2)	Multiple
	The multiple inheritance is a feature of C++.where a class can inherit from more than one class i.e. one sub class is inherited from more than one base class.
	In this type of inheritance a single child class can have multiple parent classes.
	#include <iostream>
	using namespace std;
	
	class X {
	
	public:
	
	X(){
	
	cout<<"Constructor of X class"<<endl;
		4.5		

	}
	
	};
	
	class Y {
	
	public:
	
	Y(){
	
	cout<<"Constructor of Y class"<<endl;
	
	}
	
	};			

VIII(a)	class Base
	int x;
	public:
	// parameterized constructor
	Base(int i)
	{
	x = i;
	cout << "Base Parameterized Constructor\n";
	}
	};
	class Derived : public Base
	{
	int y;
	public:
	// parameterized constructor
	Derived(int j):Base(j)
	{ y = j; cout << "Derived Parameterized Constructor\n";
	
	}
	
	};
	
		6	6	15

VIII(b)	#include<iostream>
	#include<conio.h>
	using namespace std;
	class String
	char str[20]; //member variable for string input
	public:
	void input() //member function
		9	9	

	{
	cout<<"Enter your string: ";
	cin.getline(str,20);
	}
	void display() //member function for output
	{
	cout<<"String: "<<str;
	}
	String operator+(String s) //overloading
	{ String obj;
	strcat(str,s.str);
	strcpy(obj.str,str);
	
	return obj;
	}
	};
	void main()
	{
	String str1,str2,str3; //creating three object
	str1.input();
	str2.input();
	str3=str1+str2;
	str3.display(); //displaying			

IX(a)	Virtual function is the member function of a class that can be overriden in its derived class. It is declared with virtual keyword. Virtual function call is resolved at run-time (dynamic binding) whereas the non-virtual member functions are resolved at compile time (static binding). To create virtual function, precede the function declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual to suit its own needs. Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism. #include using namespace std; class b { public:	9	9	15

	virtual void show()
	{
	cout<<"\n Showing base class....";}
	
	void display()
	{
	cout<<"\n Displaying base class...." ;
	}}
	class d:public b
	{
	public:
	void display()
	{
	cout<<"\n Displaying derived class....";
	}
	void show()
	{
	cout<<"\n Showing derived class....";
	}
	int main()
	b B;
	
	b *ptr;
	cout<<"\n\t P points to base:\n" ; ptr=&B; ptr->display();
	ptr->show();
	cout<<"\n\n\t P points to drive:\n"; d D; ptr=&D; ptr->display();
	ptr->show();
	cout<<"\n\n\t P points to drive:\n"; d D; ptr=&D; ptr->display();
	
	ptr->show();
	}			

IX(b)	C++ template allows to define the generic classes and generic functions and thus provides support for generic programming. Template is used in situation where we need to write the same function for different data types. For example, if we need a function to add two variables. The variable can be integer, float or double. For this purpose we have to write one function for each data type. To avoid writing the same function for different data types we use template. There are two types of templates in C++ : Function template Class template **Function Templates:** Function Template is just like a normal function, but the only difference is while normal function can work only on one data type	6	6	

	and a function template code can work on multiple data types.
	The general syntax of the function template is:
	template<class T>
	T function_name(T args){
	//function body
	}
	Here, T is the template argument that accepts different data types and class is a keyword. Instead of the keyword class, we can also write 'typename'.
	
	Class Templates
	Like in function templates, we might have a requirement to have a class that is similar to all other aspects but only different data types. In this situation, we can have different classes for different data types or different implementation for different data types in the same class.
	Template class also behaves similar to function templates. We need to pass data type as a parameter to the class while creating objects or calling member functions.
	template <class T>
	
	class className{
	
	public:
	T memVar;
	T memFunction(T args);
	};
	In the above definition, T acts as a placeholder for the data type. The public members' memVar and memFunction also use T as a placeholder for data types.
	Once a template class is defined as above, we can create class objects as follows:
	className<int> classObejct1;
	className<float> classObject2;
	className<char> classObject3;			

X(a)	A pure virtual function is a function which has no definition in the base class. Its definition lies only in the derived class i.e it is compulsory for the derived class to provide definition of a pure virtual function. Since there is no definition in the base class, these functions can be equated to zero. They start with **virtual** keyword and ends with = 0. The syntax for a pure virtual function. virtual void f() = 0; Example.	8	8	15

	class Base
	{
	public:
	}
	class Derived:public Base
	{
	public:
	void show()
	{
	cout << "Implementation of Virtual Function in Derived class\n"; \}
	int main()
	{
	Base *b;
	Derived d;
	
	b = &d;			

X(b)	Exceptions allow a method to react to exceptional circumstances and errors (like runtime errors) within programs by transferring control to special functions called handlers.
	
	C++ consists of 3 keywords for handling the exception. They are
	try: Try block consists of the code that may generate exception. Exception are thrown from inside the try block.
	throw: Throw keyword is used to throw an exception encountered inside try block. After the exception is thrown, the control is transferred to catch block.
	Catch: Catch block catches the exception thrown by throw statement from try block. Then,
	Syntax
	try
	{
	statements;
	throw exception;
	
	}
	catch (type argument)
	{
	statements;
	... ...
	
	}
		7	7	