2018

Group A

1a) Define access modes within a class. How can we access the class member? list out the characteristics of friend function. 2+2+2

=> Access modifiers are used to implement an important aspect of Object-Oriented programming known as Data-hiding.

There are 3 types of access modifiers available in C++:

i) Public
ii) Private
iii) Protected

i) Public -> All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too.

ii) Private -> The class members declared as private can be accessed only by the member functions inside the class. They are not allowed to be accessed directly ~~the~~ by ~~the~~ any object or function outside the class. Only the member function or the friend functions are allowed to access the private data members of a class.

CamScanner

iii) Protected : Protected access ~~specifier~~ modifier is ~~similar~~ similar to private access modifier in the sense that it can't be accessed outside of it's class unless with the help of friend class, the difference is that the class members declared as Protected can be accessed by any subclass (Derived class) of that class as well.

Syntax :-

i) public :

```
class X{
        public:
        void display();
};
```

ii) private :

```
class x{
        private:
        int a;
};
```

iii) ~~Sp~~ Protected :

```
class X{
        protected
        int b;
};
```

Date
Page

(*) characteristics of friend function

- The method and the class to which it has been declared as a friend need not be the same.
- Since it is not in the respective class's scope, it cannot be called by using its object.
- It can ~~be~~ also be called just like a normal method, even without the usage of the object.
- It can only directly access the member names by using its object name and dot membership operator along with its member name.
- There is no restriction as it is allowed to be declared in either the private or the public part.

CamScanner

b) Write a program to demonstrate constructor overloading

```
#include <iostream>
using namespace std;
class X {
        int a,b;
        public:
          X()
          {
             a = 10;
             b = 20;
          }
          X(int c, int d)
          {
             a=c;
             b=d;
          }
          void output()
          {
             cout<<"\n Default constructor\n";
             cout<<a*b;
          }
};
int main()
{
        X obj(5,60);
        X obj1;
        obj1.output();
        obj.output();
        return 0;
}
```

2a Define operator overloading with general rules. Explain types conversion and write a program for overloading the assignment operator.

=) ~~Operator~~ Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

Eg:-

Box operator + (const Box&)

operator → keyword

+ → operator symbol

The rules of operator overloading are:-

- Only built-in operators can be overloaded. If some operators are not present in C++, we cannot overload them.
- The arity of the operators cannot be changed
- The precedence of the operators remain same.
- The overloaded operator cannot hold the default parameters except function call operator "( )".

- we cannot overload operators for built-in data types. At least one user defined data types must be there.

- The assignment "=", subscript "[ ]", function call "( )" and arrow operator "->" these operators must be defined as member functions, not the friend functions.

- Some operators like assignment "=", address "&" and comma "," are by default overloaded

~~There are 3 types~~
There are two type conversion:
i) Implicit type conversion
-) Implicit type conversion is the automatic type conversion done by the compiler itself.

ii) Explicit type conversion
-) Explicit type conversion is the manual type conversion done by the programmer here the data type or the variable can be kept under function ~~and~~ " ( )" and converted one data type to the another data type.

// WAP for overloading assignment operator

```
#include<iostream>
using namespace std;

class Distance {
        private:
          int feet;
          int inches;
        public:
          Distance () {
                    feet =0;
                    inches =0;
          }
        Distance (int f, int i) {
                    feet =f;
                    inches= i;
        }
        void operator =( const Distance &D) {
                    feet = D.feet;
                    inches = D.inches;
        }
        void display Distance () {
        cout << " F :" << feet << " I: " << inches << endl;
        }
};
```

```
int main() {
    Distance D1(11,10), D2(5,11);

    cout << " First Distance :";
    D1.displayDistance();
    cout << " second Distance :";
    D2.displayDistance();

    D1=D2;
    cout << " First Distance :";
    D1.displayDistance();

    return 0;
}
```

5a) What are major and essential feature that make program object Oriented. If a base and derived class each include a member function with same name, which member function will be called by an object of a derived class, and why? Explain with an example. 1+5.

=) The major and essential feature that makes program object oriented is inheritance. - Inheritance has the ability to create classes that share the attributes and methods of existing classes, but with more specific features. Inheritance is mainly used for code reusability

```
//code
#include <iostream>
using namespace std;
class Base {
        int x,y;
        public:
        Base ()
        {
            x=20;
            y = 30;
        }
        void output ()
        {
            cout<<" The integer values are :"<<x<< y<<
                endl;
        }
};
```

```
class Derived : public Base {
        int a,b;
        public:
        Derived ()
        {
                a=25;
                b=35;
        }
        void output ()
        {
                cout<<"\n The intger values
                        are :"<<a<<  b<<endl;
        }
};
int main()
{
        Derived obj;
        obj. output ();
        obj. output();
        return 0;
}
```

If we call a member function having same name in both base and derived class then the memberfunction of derived class will be called because the object is made from the derived class and if the object contains its own memberfunction so at first it will call its own memberfunction.

5) What is abstract class? Write a program in C++ that implements the concept of multilevel inheritance.

=> An abstract class in C++ is one that has at least one pure virtual function by defination. In other word, a function that has no definition. The abstract class's descendents must define the pure virtual function; otherwise, the subclass would become an abstract class in its own right.

```
// code
#include<iostream>
using namespace std;
class X {
        int value1;
        public:
        X(int a)
        {
                value1=a;
        }
        void output1()
        {
                cout<<"\n The value in the
                        base class is:"<<value1;
        }
};
```

```
class Y : public X {
        int value2;
        public :
        Y (int a , int b) :X(a)
        {
            value2=b;
        }
        void output2()
        {
            cout<<"\n The value in the
                subclass is: "<< value2;
        }
};
class Z: public Y {
        int value3;
        public:
        Z (int a , int b, int c) : Y(a,b)
        {
            value3=c;
        }
        void output3()
        {
            cout <<"\n The value of the
                another subclass is:"<<value3;
        }
};
int main()
{
        Z obj3(1,2,3);
        obj3. output1();
        obj3. output2();
        obj3 . output3();
        return 0;
}
```

Group 'B'

4. Discuss access specifier. Describe static data members and static member functions with a suitable program.

=) Access modifiers are used to implement an important aspect of Object-Oriented programming known as Data hiding. Access modifiers or Access specifiers in a class are used to assign the accessibility to the class members. That is, its sets some restrictions on the class members not to get directly accessed by the outside functions.

Static data member are class members that are declared using static keywords. The static data member is always initialized to zero when the first class object is created.

A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator::. A static member function can only access static data member, other static member functions and any other functions from outside the class.

```
// code
#include<iostream>
using namespace std;
class notebook{
        static int page_number;
        public:
          static int value()
          {
                return page_number;
          }
};
int notebook::page_number=1;
int main()
{
        cout<<"Numbers="<<notebook::value()
          <<endl;
        return 0;
}
```

5. Define constant pointer and pointer to constant with a suitable example. Write a program to demonstrate conversion from one class to another class type without using constructor.

=) Constant pointer in C++ if one has a value in his/her program and it should not change throughout the program, or if one has a pointer and he/she doesn't want it to be pointed to a different value, he/she should make it a constant with the const keyword.

```
//code
#include <iostream>
using namespace std;

int main()
{
    int a=10;
    int *const i=&a;
    cout<<" 1st pointed value :"<<*i;
    *i=99;
    cout<<"\n Modified value :"<<*i;
    return 0;
}
```

pointer to constant in c++

A pointer to constant is a pointer through which the value of the variable that the pointer points cannot be changed. The address of these pointers ~~cannot~~ can be changed, but the value of the variable that the pointer points cannot be changed. Syntax of pointer to constant

const <type of pointer>* <name of pointer>

```
//code
#include <iostream>
using namespace std;
int main()
{
        int num=10;
        const int *ptr_const;

        ptr_const = &num;

        num=20;

        *ptr_const =100;

        printf ("Num = %d\n", num);
        printf ("Num = %d\n", *ptr_const);

        return 0;
}
```

```
//code for ~~eat~~ class to another class
#include<iostream>
using namespace std;
class Rectangle {
        int length, width, area;
        public:
        Rectangle (int l, int w)
        {
            length = l;
            width = w;
            area = l * w;
        }
        void display()
        {
            cout<<"\n The area of the
                rectangle is "<< area;
        }
};
class Triangle {
        int base, height;
        float area1;
        public:
        Triangle (int b, int h)
        {
            base = b;
            height = h;
            area1 = 0.5* base* height;
        }
        void print ()
        {
            cout<<"\n The area of the triangle is:"
                << area1;
        }
```

```
    operator Rectangle()
    {
        Rectangle temp (base, height);
        return temp;
    }
};
int main()
{
    Triangle t(10,20);
    Rectangle r=t;
    t.print();
    r.display();
    return 0;
}
```

6. Discuss a default argument. Write a program to find perimeter of a square, rectangle, and circle using function overloading. 3+5

=) A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument with a ~~default~~ default value. In case any value is passed the default argument is overridden.

```
// code
#include<iostream>
using namespace std;

int sum (int x, int y, int z=0, int w=0)
{
        return (x+y+z+w);
}
int main()
{
        cout << sum (10,15) << endl;
        cout << sum (10,15,25) << endl;
        cout << sum( 10,15,25,30 )<< endl;
        return 0;
}

// code for function overloading
#include<iostream>
using namespace std;
int peri (int u)
{
        int a;
        a= u*x;
        return a;
}
int peri( int f, int g)
{
        int b,c;
        c = f+g;
        b= 2*c;
        return b;
}
```

```
float peri (float r)
{
        float d;
        float pie = 3.14;
        d = 2* pie *r;
        return d;
}
int main()
{
        int len, bre, hei;
        float radius;
        cout << "Enter the length :";
        cin>> len;
        cout << "\n Enter the breadth:";
        cin>> bre;
        cout << "\n Enter the radius:";
        cin>> radius;
        cout << "The perimeter of square is: " <<
                peri( len) << endl;
        cout << "\n The perimeter of the rectangle
                is " << peri (len, bre) << endl;
        cout << "\n The perimeter of the circle is "
                << peri (radius ) << endl;
        return 0;
}
```

7. Explain polymorphism and compare compile time and run time polymorphism. Describe virtual and pure virtual function with an example. 8.

=) The word polymorphism means having many forms. Typically, polymorphism occurs when there is hierarchy of classes and they are related by inheritance.
C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Comparision between compile time and run-time polymorphism

	Compile time polymorphism	Run-time polymorphism
1)	In compile time polymorphism the call is resolved by the compiler.	In run-time polymorphism, the call is not resolved by the compiler
2)	It is also known as static binding, early binding and overloading as well	It is also known as Dynamic binding, late binding and overriding as well.
3)	It is achieved by function overloading and operator overloading	It is achieved by virtual functions and pointers.

4	It provides fast execution because the method that needs to be executed is known early at the compile time.	It provides slow execution as compare to early binding because the method that needs to be executed is known at the run time.
5.	Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run-time.

Virtual function - A virtual function is a function in a base class that is declared using the keyword virtual.

```
//code
# ~~iostream~~ include<iostream>
using namespace std;
class base {
public:
        virtual void print()
        {
            cout<<" print base class\n";
        }
        void show()
        {
            cout<<" show base class\n";
        }
};
```

```
class derived : public base {
        public:
        void print()
        {
            cout<<"print derived class\n";
        }
        void show()
        {
            cout<<"show derived class\n";
        }
};
int main()
{
        base *bptr;
        derived d;
        bptr = &d;

        // virtual function, binded at run time
        bptr -> print();

        // Non-virtual function, binded at compile time
        bptr -> show();
        return 0;
}
```

pure virtual function -> A pure virtual function is a virtual function in c++ for which we need not to write any function definition and only we have to declare it. It is declared by assigning 0 in the declaration.

```
// code
#include<iostream>
using namespace std;

class Base
{
        int n;
        public:
          virtual void fun()=0;
          int getX()
          {
                return n;
          }
};
class Derived : public Base
{
        int y;
        public:
        void func()
        {
                cout << "func() called";
        }
};
```

```
int main(void)
{
        Derived d;
        d.fun();
        return 0;
}
```

9. Define template. Write a program using function template to find the sum of first and last element of an array of size, N of type int and float.

=) Template are the fundation of generic programming, which involves writing code in a way that is independent of any particular type.

* template syntax

```
template <classtype> ret_type func-name (parameter list) {
        // body of function
}
```

```
//code
#include<iostream>
using namespace std;
template <typename R>
void sum(R a[], int y)
{
    cout<<"\n The sum of first and last
        element in the array is :"
        <<a[0]+a[y-1];
}
int main()
{
    int b=0, c[30], n, m;
    float d[30];
    cout<<"Enter the number of elements
    you want to enter in integer :";
    cin>>n;
    cout<<"Enter the elements :"<<endl;
    for(b=0; b<n; b++)
    {
        cin>>c[b];
    }
    sum<int>(c, n);
    cout<<"\n\n Enter the number of
        elements you want to enter
        in float :";
    cin>>m;
    cout<<"Enter the elements :"<<endl;
    for(b=0; b<m; b++)
    {
        cin>>d[b];
    }
    sum<float>(d, m);
    return 0;
}
```

10. What are the advantages of the exception handling mechanism? Explain the components of ~~exc~~ exception handling mechanism with suitable example.

=> The advantages of the exception handling mechanism are:

i) Separation of Error handling code from Normal Code:

-> In traditional error handling codes, there are always if else conditions to handle errors. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try catch blocks, the code for error handling becomes separate from the normal flow.

ii) Functions / Methods can handle any exceptions they choose: A function can throw many exceptions, but may choose to handle some of them. The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by called of the caller. In c++ a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it)

iii) Grouping of error Types : In C++, both basic types and objects can be thrown as exception, we can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.

The component of exception handling are

-> throw - A program throws an exception when a problem shows up. This is done using a throw keyword.

-> catch - A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception

-> try - A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

```
//code
#include <iostream>
using namespace std;

double division (int a, int b)
{
        if (b==0)
        {
            throw "Division by zero condition!";
        }
        return (a/b);
}
int main()
{
        int n=50;
        int y=0;
        double z=0;
        try{
            z= division (n,y);
            cout << z << endl;
        }
        catch (const char* msg)
        {
            cerr << msg << endl;
        }
        return 0;
}
```

11. Write short notes on any TWO:

a) Abstract class

-) An abstract class is a class that is declared with an abstract keyword which is a restricted class hence cannot be used to create objects; however they can be subclassed. To access ~~an~~ abstract class, it must be inherited from another class.

An abstract class in C++ is one that has at least one pure virtual function by defination. In other words, a function that has no ~~g~~ defination. The abstract class's descendents must define the pure virtual function; otherwise, the subclass would become an abstract class in its own right.

```
//code
#include<iostream>
using namespace std;
class Base{
        public:
            void fun1()
            {
                cout<<"Base fun1:";
            }
            virtual void fun2()=0;
};
```

```
class Derived : public Base
{
        public:
            void fun2()
            {
                cout<<"Derived fun2!";
            }
};
int main()
{
        Base *p = new Derived();
        p -> fun2();
        return 0;
}
```

5) Array in C++

=) An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier. That means that, for example, five value of type int can be declared as an array without having to declare 5 different variables (each with its own identifiers).

Array are used to store multiple values in a single variable, instead of declaring ~~spe~~ separate variables for each value.
To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the number of elements it should store:

```
string cars[4];    // declared
```

c) New and delete operator

-) New operator is used to allocate memory location during program execution. This operator allows allocates memory for the specified object and returns its memory address (i.e. starting address of the allocated memory) to the pointer variable. The allocated memory has n name. the data into the allocated memory location is accessed with the reference of the pointer. The new operator can be used to allocate memory for simple variables, arrays, or user-defined objects etc

Syntax:

ptr = new data_type

Example;

```
Float * p;
p = new float;
```

Delete operator = Once memory is allocated with new operator, it cannot be de-allocated automatically. If the pointer (that points to the allocated memory) goes out scope, ~~the~~ the allocated memory becomes unavailable.

The delete operator is used to de-allocated memory occupied by an object. The de-allocated memory can be used by operating system for other purposes

Syntax:

Delete [ ] ptr;

where

ptr represents the pointer variable which holds the memory address of the object which is to deleted from memory.

2019

Group 'A'

1a) How is ~~di~~ data hiding accomplished? (3)

=> Data hiding can be accomplished by the use of ~~p~~ access modifiers in C++. The access specifiers or modifiers are used to assign the accessibility to the class members. It sets some restrictions on the class members not to get directly accessed by the outside function.

There are 3 types of access specifiers in C++

i) Public
ii) Private
iii) Protected.

Explaination

i) public access specifier

-) In the public access specifier the data member and the member function inside the class ~~ar~~ can be publicly accessed ~~f~~ from outside the class. It can be directly accessed by the object of the class.

Syntax:-

```
class X {
        public:      // access specifier

            int a=10;   // public data member
};
```

ii) private data access specifier

=) By the use of private access specifier the data member and member function becomes private and can't be accessed directly. Mainly, the private access specifier is used for the data member. As it can't be accessed directly the other member function which should be public will help to place the value in the private data member.

Syntax:-

```
class Apple{
        private:        //access specifier
        int n;          // data member (private)
        public:         // public access specifier
        void get_details(int a)
        {               // public memberfunc
            n = a;
        }
};
```

iii) Protected ~~data member~~ access specifier

=) The data member and member function accessed ~~by~~ under the protected access specifier is same as the private access specifier they can also not be directly accessed by the object but the protected access specifier is a bit different from the private access specifier. The protected data member or member function can be

accessed by the help of derived class / sub-class or friend function.

Syntax:

```
class X{
        protected: //protected ~~dar~~ access specifier
        int n; //protected data member
        public:
        friend void get_details();
};
```

5) Differentiate between object-Oriented paradigm and procedural oriented program. (4)

=)

Procedural Oriented Program	Object-Oriented program
1) Program is divided into small parts called function.	program is divided into parts is called objects.
2) Importance is not given to data but to function as well as sequence of actions	Importance is given to the data rather than procedures or functions because it works as a real world.
3) It follows Top-down approach	It follows Bottom-up approach
4) It doesn't have access specifier	It has ~~&~~ 3 access specifier
5) It doesn't have any proper way for hiding data so it is less secure	It provides data hiding so provides more security
6) Eg:- C, FORTRAN, PASCAL etc	Eg:- C++, JAVA, .NET etc

c) Discuss implicit and explicit type conversion with proper examples. (5)

→ Implicit type conversion is the automatic type conversion done by the compiler itself.

```
// Implicit type conversion code
#include<iostream>
using namespace std;

int main()
{
        int num_int = 27;
        double num_double = 0;

        num_double = num_int;

        cout<<"Integer value :"<< num_int ;
        cout<<"\n Double value :" << num_double;

        return 0;
}
```

Explicit type conversion is the type conversion where the user manually changes data from one type to another. ~~This is known as~~

There are 3 types of explicit type conversion

i) C-style type casting

-> This C-style type casting is also known as ~~type~~ cast notation. This type of casting is favoured by the C programming language

```
#include<iostream>
using namespace std;

int main()
{
        int num_int = 32;
        double num_double;

        num_double =(double) num_int;

        cout<<"Integer value:" << num_int;
        cout<<"\n Double value:"<< num_double;

        return 0;
}
```

ii) Function style casting

-> In this type casting we can also use the function like notation to cast data from one type to another

```
#include <iostream>
using namespace std;

int main()
{
        int num_int = 33;
        double  num_double = 0;

        num_double = double (num_int);

        cout<<"Integer value : "<< num_int;
        cout<<"\n Double value : "<< num_double;

        return 0;

}
```

iii) Type casting

=) Type casting is the combination of both c-style casting and function-style.

```
#include <iostream>
using namespace std;

int main()
{
        double num_double = 3.56;
        cout << " num_double = " << num_double ;

        int num_int1 = (int )num_double;
        cout << " num_int1 = " << num_int;

        int num_int2 = int (num_double);
        cout << " num_int2 = " << num_int2 << endl;

        return 0;
}
```

2a) Discuss static data member and static member function with proper illustration

=> static data member are class member that are declared using static keyword. The static data member is always initialized to zero when the first class object is created.

A static member function can be called even if no objects of the class exists and the static function are accessed using only the class name and the scope resolution operator ::.
A static member function can only access static data member, other static member functions and any other functions from ~~of~~ outside the class.

```
//code
#include <iostream>
using namespaces std;
class notebook {
        static int page_num;
        public:
          static int value()
          {
                return page_num;
          }
};
int notebook ::page_num=2;
int main()
{
        cout<<" Number= " << notebook::value()<<endl;
        return 0;
}
```

5) Write a program to illustrate copy constructor

```
#include <iostream>
using namespace std;
class Demo {
        private:
         int num1, num2;
         public:
         Demo (int n1, int n2)
         {
                num1=n1;
                num2=n2;
          }
          Demo (const Demo &n)
          {
                num1=n.num1;
                num2=n.num2;
          }
          void display()
          {
                cout << " num1= " << num1 << endl;
                cout << "num2= " << num2 << endl;
           }
};
int main();
{
        Demo obj1 (25, 75);
        Demo obj2= obj1
        obj1.display();
        obj2.display();
        return 0;
}
```

c) Write a program to add two complex numbers of two different classes. 6

```
#include<iostream>
#include<conio.h>
using namespace std;
class secondComplex;
class firstComplex{
        public:
            int real;
            int img;
            friend class secondComplex;
};
class secondComplex{
        public:
            int real;
            int img;
            friend void add();
};
void add()
{
        firstComplex obj1;
        SecondComplex obj2;
        int r,s;
        cout<<"Enter real and imaginary for
            the first complex number below:"<<endl;
        cout<<"FirstReal=";
        cin>>obj1.real;
        cout<<"firstImaginary=";
        cin>>obj1.img;
```

```
cout<<"Enter the real and imaginary
        for the second complex number
        below:"<<endl;
cout<<"secondReal=";
cin>>obj2.real;
cout<<"secondImaginary=";
cin>>obj2.img;
r=obj1.real+obj2.real;
s=obj1.img+obj2.img;
cout<<"sum of two complex
        number="<<r<<"+"<<s<<"i";
getch();
}
int main()
{
        add();
        return 0;
}
```

3a) Give a suitable example for default argument.

```
#include <iostream>
using namespace std;
int sum (int a=0, int b=20, int c=30);
int main()
{
        sum(1);
        sum(1,2);
        sum(1,2,3);
        return 0;
}
int sum (int a, int b, int c)
{
        int z;
        z=a+b+c;
        cout<<"\n The sum of the integer are:"
                <<z;
}
```

3b) Create a class named Employee with four data members (ID, name, position, and salary). Using member functions initialize the data members for 3 objects and display information of all three of them. Assume appropriate data types.

```
//code
#include<iostream>
using namespace std;
class employee {
        int id;
        string name;
        string position;
      int salary;
      public:
      void set_details (int a, string b, string c,
                        ~~int~~ int d)
      {
            id= a;
            name =b;
             Position =c;
             Salary=d;
      }
      void display (employee X[3])
      {
            int i;
            for (i=0, i<3, i++)
            {
                  cout<<"The id of the employee:"<<X[i].id;
                  cout<<"\n The name of the employee:
                        "<<X[i].name;
                  cout<<"\n The position of the
                        employee:" << X[i].position;
                  cout<<"\n The salary of the
                        employee:"<<X[i].salary;
            }
      }
};
```

```
int main()
{
    employe obj[3], temp[3];
    string e,f;
    int , i, j, k;
    for (i=0; i< 3; i++)
    {
        cout<<" Enter the id of the "<<i+1<<
            " Employee :";
        cin >>j;
        fflush (stdin);
        cout<< " Enter the name of the "<<i+1<<
        " Employee :",
        getline (cin, e);
        fflush (stdin);
        cout<<" Enter the position of the "
            <<i+1<<" Employee :";
        getline (cin, f);
        fflush(stdin);
        cout<< "Enter the salary of the
            "<<i+1<<" Employee :";
        cin>>k;
        fflush(stdin),
        obj[i].get_details(j,e,f,k);
    }
    temp[3].display (obj);
    return 0;
}
```

Group B

4. Write a program using function template that ask user to enter the 10 elements in the array of type int and float and display the five largest values in ascending.

```
#include <iostream>
#include <iomanip.h>
using namespace std;
template <typename R>
void ascen (R a[ ])
{
        int i,j;
        R temp;
        for (i=0 ;i<10;i++)
        {
                for (j=i+1 , j<10;j++)
                {
                        if (a[i] < a[j])
                        {
                                temp=a[i];
                                a[i] =a[j];
                                a[j] =temp;
                        }
                }
        }
Cout << "The ascending order of the
        largest number are:"<<endl;
```

```
        for (i=0; i<s; i++)
        {
            cout << a[i] << endl;
        }
}
int main()
{
        int a[10], k, m;
        float af[10];
        label:
        cout << "Enter the numbers you want in
            ascending order:" << endl;
        cout << "1. Integer " << endl;
        cout << "2. Float " << endl;
        cin >> m;
        if (m==1)
        {
            cout << "\n Enter the numbers:" << endl;
            for(k=0; k<10; k++)
            {
                cin >> a[k];
            }
            ascen <int>(a);
        }
        else if (m==2)
        {
            cout << "\n Enter the numbers:" << endl;
            for (k=0; k<10; k++)
            {
                cin >> af[k];
            }
            ascen <float>(af);
        }
```

```
        else
        {
            cout<<"\n Invalid choice !! !";
            goto label1;
        }
        return 0;
}
```

5. Write a program to overload binary '+' operator

```
//code
#include<iostream>
using namespace std;
class Distance{
    public:
        int feet, inch;
        Distance()
        {
            this -> feet=0;
            this -> inch=0;
        }
        Distance (int f, int i)
        {
            this -> feet = f;
            this -> inch = i;
        }
```

```
        Distance operator +(Distance &d2)
        {
            Distance d3;
            d3.feet = feet + d2.feet;
            d3.inch = inch + d2.inch;
            return d3;
        }
};
int main()
{
    Distance d1(8,9);
    Distance d2(10,2);
    Distance d3;
    d3 = d1+d2;
    cout << "\nTotal feet :" << d3.feet << "\nTotal
        inch :" << d3.inch;
    return 0;
}
```

6. Define function prototype, How function overloading is achieved? Mention advantages of using an inline function.

=> Function prototype is basically a declaration of the function that tells the program about the type of the value which is to be returned by the function. It also tells about the number and type of arguments of the function. Function prototyping is one of the very useful features in C++ as it enables the compiler to perform more powerful checking.

Function overloading is achieved by creating multiple version of a function in a class, by the following rules, such as - All versions of an overloaded function must have the same name. All versions of an overloaded function must have different arguments.

//code

Advantages of inline function are:-

i) Function call overhead doesn't occur

ii) It also saves the overhead of push/pop variables on the stack when function is called.

iii) It also saves overhead of a return call from a function

iv) When you inline a function, you may enable compiler to perform context specific optimization on the body of function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of calling context and the called context.

v) Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return.

7. Define pure virtual function and abstract class. Write a program to demonstrate ambiguity in multiple inheritance.

=) A pure virtual function is a virtual function in C++ for which we need not to write any function definition and only we have to declare it. It is declared by assigning 0 in the ~~dec~~ declaration.

An abstract class in C++ is a class which have at least one pure virtual function. An abstract class can have normal functions and variables along with a pure virtual function. It is mainly used for upcasting so that its derived classes can use its interface. The abstract class descendants must define the pure virtual function; otherwise, the subclass would become an abstract class in its own right.

// code

```
#include<iostream>
using namespace std;

class A {
        public:
        Father()
          {
                cout << "He is a father." << endl;
          }
};
```

```
class B{
    public:
    Mother()
    {
        cout << " she is a mother." << endl;
    }
};
class Son : public A, public B{
};
int main()
{
    Son b1;
    b1.Father(); /* causes Ambiguity*/
    return 0;
}

/* Ambiguity resolution
int main()
{
    son b;
    b.A :: Father();
    b.B:: Father();
    return 0;
}*/
```

8. Create a class student that stores name and roll. From this class, derive a class marks that stores marks for 3 subjects. Then from the class marks derive a class record which stores semester and average marks for 3 subjects. Create an object for class

record and display name, ~~roo~~ roll, marks in 3 subjects, semester and average marks for a student. Assume appropriate data types 8

```
# include<iostream>
# include<conio.h>
using namespace std;
class student {
        public:
        string name;
        int roll;
};
class marks : public student{
        public:
        int mar1;
        int mar2;
        int mar3;
};
class record : public marks{
        public:
        int semester;
        int avg1;
        void average ()
        {
                int avg;
                avg = mar1+mar2+mar3;
                avg1 = avg/3;
        }
        void display()
        {
```

```
        cout<<"\n The name of the student:"<<name;
        cout<<"\n The roll of the student:"<<roll;
        cout<<"\n The semester of the student:"<<
                semester;
        cout<<"\n The marks of the three
                subject are:"<<endl;
        cout<<mar1<<endl;
        cout<<mar2<<endl;
        cout<<mar3<<endl;
        cout<<"\n The average marks of the
                student:"<<avg1;
    }
};
int main()
{
        record std;
        cout<<"Enter the name of the student:";
        cin>>std.name;
        fflush(stdin);
        cout<<"\n Enter the roll no of the student:";
        cin>>std.roll;
        fflush(stdin);
        cout<<"\n Enter the marks of the
                three subjects respectively:"<<endl;
        cin>>std.mar1>>std.mar2>>std.mar3;
        fflush(stdin);
        cout<<"\n Enter the semester of the
                student:";
        cin>>std.semester;
        fflush(stdin);
        std.average();
        std.display();
        return 0;
}
```

10. Mention features of this pointer. Write a program to demonstrate use of ~~constructor~~ constructor in derived class. 2+6

=) Features of this pointer are :-

i) To assign value of the function's parameter to a class member variable correctly

ii) In a large code base of a function, identifying the class member variable is easy.

iii) Internal use of this pointer as an argument to a function

iv) The 'this' pointer is passed as a hidden argument to all nonstatic member function calls and is available as a local variable within the body of all nonstatic functions.

```
//code
#include<iostream>
using namespace std;
class BASE {
        int value1;
        public:
          BASE (int x)
          {
                value1=x;
          }
          void display()
          {
                cout<<"\n The value of BASE
                      class is:"<<value1;
          }
};
```

```
class DERIVED : public BASE{
        int value2;
        public:
         DERIVED (int x, int y): BASE(x)
        {
                value2 = y;
        }
        void print()
        {
                cout<<"\nThe value of DERIVED
                        class is: "<<value2;
        }
};
int main()
{
        DERIVED obj1 (1,2);
        obj1.display();
        obj1.print();
        return 0;
}
```

11. Write short notes on any Two.

a) Namespaces

=) A namespace defination begins with the keyword namespace followed by the namespace name. Namespace allow us to group named entities that otherwise would have global scope into narrower scopes, giving them ~~namescope~~ namespace scope.

A namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc) inside it. Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scops.

Syntax

```
namespace   namespace_name
{
        int x,y;
}
```

3) Exception handling

=) An exception is a problem that arises during the execution of a program. A c++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. c++ exception handling is built upon three keywords : try, catch and throw.

- throw - A program throws an exception when a problem shows up. This is done using a throw keyword.

- catch - A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

- try - A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

2021

1a) Write a program to create a class named Employee_Info with data member Emp_Id and Emp_Name. Create another class named Finance_Info from Employee_Info with data member Bas_Sal and ot_sal. Create an independent class Extra_Allowances with data member Tray_All, House_All and Dail_All. Now, derive another class named Total_Info from Finance_Info and Extra_Allowances with data member Tot_Sal. Make necessary function for the above and show the Gross salary.

```
//code
#include<iostream>
using namespace std;
class Employee_Info{
        protected:
                int Emp_Id;
                string Emp_name;
                public:
                    void get_details (int o, string p)
                    {
                        Emp_Id = o;
                        Emp_name = p;
                    }
};
class Finance_Info: public Employee_Info{
        protected:
                int Bas_Sal;
                int Ot_Sal;
                public:
```

```
        void details (int r , int s)
        {
                Bas_Sal=r;
                Ot_sal =s;
        }
};
class Extra_Allowances {
        protected:
            int Tray_All;
            int House_All;
            int Dail_All;
            public:
                void det (int t, int h, int d)
                {
                    Tray_All =t;
                    House_All=h;
                    Dail_All =d;
                }
};
class Total_Info : public Finance_Info, public
  Extra_Allowances {
        public:
            int Tot_sal;
            void grossSal()
            {
                int pf;
                pf = Bas_Sal* 0.11;
                Tot_Sal = Bas_sal + Ot_sal +
                    House_All + Tray_All + Dail_All
                    - pf;
            }
```

```
void display()
{
        cout<<"\n The Id of the
                employee : "<< Emp_Id;
        cout<<"\n The Name of the
                employee :"<< Emp_name;
        cout<<"\n The Gross salary of the
                employee :" << Tot_Sal;
}
};
int main()
{
        Total_info obj1;
        int id, bs, ots, trA, hoA, daA;
        string n;
        cout<<"\n Enter the Id of the employee :";
        cin>>id;
        fflush(stdin);
        cout<<"\n Enter the name of the employee:";
        getline(cin,n);
        fflush(stdin);
        obj1.get_details (id,n);
        cout<<"\n Enter the Basic Salary of the
                employee :";
        cin>>bs;
        fflush(stdin);
        cout<<"\n Enter the Outstanding Salary of the
                employee :";
        cin>> ots;
        fflush(stdin);
        obj1.details (bs,ots);
```

```
cout<<"\n Enter the Tray Allowance of
        the employee :";
cin >> trA;
fflush(stdin);
cout<< "\n Enter the House Allowance of the
        employee :";
cin>> hoA;
fflush(stdin);
cout<<"\n Enter the Daily Allowance of the
        employee :";
cin>> daA;
fflush(stdin);
obj1.det(trA, hoA, daA);
obj1.grosssal();
obj1.display();
return 0;
}
```

5) Discuss function overloading with example

=) The function overloading in c++ feature is used to improve the readability of the code. It is used so that the programmer does not have to remember various function names. If any class has multiple functions with different parameters having the same name, they are said to be overloaded.

```
//code
#include <iostream>
using namespace std;
int are (int b, int h)
{
        int recArea;
        recArea = b*h;
        return recArea;
}
int are (int l)
{
        int squareArea;
        squareArea = l*l;
        return sequare Area;
}
float are(float r)
{
        float circleArea, radius;
        radius = r*r;
        circle Area = 3.14* radius;
        return circleArea;
}
int main()
{
        cout << "\n Area of the Rectangle :" << are (5,4);
        cout << "\n Area of the square :" << are (5);
        cout << "\n Area of the circle :" << are(4.4f);
        return 0;
}
```

2a) What is constructor and what are the properties of constructor? Write a program showing the example of parameterized constructor.

=) A constructor is a special type of member function of a class which initializes objects of a class. In C++, constructor is automatically called when object is created. It is a special member function of the class because it doesnot have any return type.

The properties of constructor are:-

i) Constructor has same name as the class itself.

ii) Default constructor don't have input argument however, copy and parameterized constructors have input arguments.

iii) Constructors don't have return type

iv) A constructor is automatically called when an object is created.

v) It must be placed in public section of class.

vi) If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).

```
//code
#include <iostream>
using namespace std;
class cons {
        int feet, inch;
        public:
            cons (int f, int i)
            {
                feet = f;
                inch = i;
            }
            void display()
            {
                cout <<"\n The value in feet is :"<<feet;
                cout <<"\n The value in inch is :"<<inch;
            }
};
int main()
{
        cons obj(10,5);
        obj.display ();
        return 0;
}
```

b) In what circumstances we need to use Exception Handling Mechanism. Discuss with Example. 4

⇒ Some codes where the data is not divisible by 0 we use the Exception handling mechanism.

i) Separation of Error handling code from Normal code - Exception handling makes the code less readable and maintainable. with try catch blocks, the code for error handling becomes separate from the normal flow.

ii) Function / Methods can handle any exceptions they choose. ~~A fun~~ A function can throw many exceptions but may choose to handle some of them. The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by the caller of the caller. In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or ~~catchi~~ catching it).

iii Grouping of Error types: In C++, both basic types and objects can be thrown as exception. We can create a hierarchy of exception objects

group exceptions in namespaces or classes, categorize them according to types.

```
// code
#include <iostream>
using namespace std;

double division (int a, int b)
{
        if (b==0)
        {
            throw "Division by zero condition!";
        }
        return (a/b);
}
int main()
{
        int n=50;
        int y=0;
        double z=0;
        try
        {
            z= division (n,y);
            cout << z <<endl;
        }
        catch (const char* msg)
        {
            cerr << msg << endl;
        }
        return 0;
}
```

3a) List different types operators which cannot be overloaded? WAP to read two strings and concatenate them showing the example of operator overloading.

=) The different types of operators which cannot be overloaded are:-

i) (.) dot operator
ii) (?:) ternary operator
iii) (::) scope resolution operator
iv) (.*) pointer to member operator
v) sizeof() operator
vi) typeid object type operator

```
//code
#include <iostream>
using namespace std;
class conca{
        string name;
        public:
        void get_details(string b)
        {
                name = b;
        }
        conca operator +(const conca & obj)
        {
                conca temp;
                temp.name = name + obj.name;
                return temp;
        }
```

```
void output()
{
    cout<<"The concatenation of the
        two string are :"<<name;
}
};
int ~~name~~ main()
{
    conca obj1, obj2, result;
    string a,c;
    cout<<"Enter the first string:";
    getline (cin,a);
    obj1.get_details(a);
    cout<<"\n Enter the second string:";
    getline (cin,c);
    obj2. get_details (c);
    result = obj1 + obj2;    // Binary operator call
    result. output();
    return 0;
}
```

5) What are ~~the~~ the different types of visibility modes used in Inheritance? Discuss.

=) The different types of visibility modes used in inheritance are public protected, private.

A derived class can access all the non-private members of its base class. Thus, base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

There are 3 modes of inheritance

i) Public Mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

ii) Protected mode: If we derive a subclass from a protected base class. Then both public members and protected members of the base class will become protected in the derived class.

iii) Private Mode: If we derive a subclass from a private base class. Then both public members and protected members of the base class will become Private in the derived class

Group B

4. What is a template and why do we use Template in OOP? WAP to swap to variables using Function Template.

=) Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blue print or formula for creating a generic class or a function. We use Template in OOP to pass data type as a parameter so that we don't need to write the same code for different data types. For example, a software company ~~may~~ may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.

```
//code
#include <iostream>
using namespace std;
template <typename R>
void swapp (R a, R b)
{
    R temp;
    temp = a;
    a = b;
    b = temp;
    cout << "\n Integers after swapping are: " << endl
        << a << endl << b;
}
```

```
int main()
{
    int x, y, result;
    cout<<"Enter the two integers you
        want to swap:"<<endl;
    cin>>x>>y;
    Swapp<int>(x,y);
}
```

5. What is inheritance? List different types of inheritance supported in c++. Write an OOP showing multiple inheritance.

=> Inheritance is one of the key features of Object-oriented programming in c++. It allows us to create a new class (derived class) from an existing class (base class).

The derived class inherits the features from the base class and can have additional features of its own.

The different types of inheritance are:-

i) single inheritance
ii) multiple inheritance
iii) multipath inheritance
iv) multilevel inheritance
v) Hierarchical inheritance
vi) Hybrid (virtual) inheritance

```
// code
#include<iostream>
using namespace std;
class KIST {
        protected:
        int stud1;
        public:
            void get_details1 (int s1)
            {
                stud1 = s1;
            }
            void display()
            {
                cout<<"\n The total number of
                student in KIST :"<<stud1;
            }
};
class TRINITY {
        protected:
        int stud2;
        public:
            void get_details2 (int s2)
            {
                stud2 = s2;
            }
            void print()
            {
                cout<<"\n The total number of
                student in Trinity :"<<stud2;
            }
};
```

```
class students : public KIST, public TRINITY{
        public:
        int outstu;
        int tot_stud;
        void totalStudents()
        {
            tot_stud = stud1 + stud2 + outstu;
        }
        void output ()
        {
            cout<<"\n Total number of student
                who read in same class are :"<<tot_stud;
        }
};
int main()
{
        student obj;
        int a,b;
        cout<<"\n Enter the total number of
            student in kist college :";
        cin>>a;
        obj. get_details1(a);
        cout<<"\n Enter the total number of
            student in trinity college :";
        cin>>b;
        obj. get_details2 (b);
        cout<<"\n Enter the students from
            outside valley :";
        cin>> obj. outstu ;
        obj. totalstudents ();
        obj. display ();
```

```
        obj.print();
        obj.output();
        return 0;
}
```

7. What are the different types of type conversion possible in operator overloading? Write a program showing the example of basic to class type conversion.

=) The different types of type conversion possible in operator overloading are:-

i) Basic to class conversion
ii) class to basic conversion
iii) Class to class conversion

```
//code
#include<iostream>
using namespace std;
class Time{
        int hrs, min;
        public:
            Time (int t)
            {
                hrs = t/60;
                min = t%60;
            }
            void display()
            {
                cout<< hrs << ":hr" <<endl;
                cout<< min<<":min" <<endl;
            }
};
```

```
int main()
{
    int duration;
    cout<<"Enter the duration in minutes:";
    cin>>duration;
    Time t1=duration;
    t1.display();
    return 0;
}
```

8. What is DMA in C++? Write a program showing the example of new and delete operator.

=> DMA in C++ allows us to allocate memory to storage components of our program at run-time. It is one of the most significant features of C++. For the allocation of the memory new operator is used.

```
//code
#include<iostream>
using namespace std;
int main()
{
    int *p1, *p2, sum;
    p1=new int;
    p2=new int;
    cout<<"Enter first value:";
    cin>>*p1;
    cout<<"Enter second value:";
    cin>>*p2;
```

```
sum = *p1 + *p2;
cout << "sum of values = " << sum << endl;
delete p1;
delete p2;
return 0;
}
```

9. What is virtual function? Write a program showing the example of virtual function.

=) Virtual member is defined ~~as the~~ in the base class and can further be defined in the child class as well.

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.

```
// code
# include <iostream>
using namespace std;
class base{
    public:
        virtual void print()
        {
            cout << " print base class";
        }
        void show()
        {
            cout << " show base class";
        }
};
```

```
class derived : public base {
        public :
                void print()
                {
                        cout << "print derived class\n";
                }
                void show()
                {
                        cout << " show derived class\n";
                }
};
int main()
{
        base *bptr;
        derived d;
        bptr = &d;
        bptr -> print();
        bptr -> show();
        return 0;
}
```

10. What are the different ways of defining members of a class? Discuss with example. How encapsulation is achieved in c++?

=> The different ways of defining members of a class are:-

i) Inside class defination

ii) Outside class defination

i) Inside class defination

1) Defining the members inside class is normal way of defining no other operator or keyword is used

Example:-

```
//code
#include <iostream>
using namespace std;
class Geeks {
        public:
        string geekname;
        void printname()   //inside class definition
        {
            cout <<"Geekname is :"<< geekname;
        }
};
int main()
{
        Geeks obj1;
        obj1.geekname = "Abhi";
        obj1.printname();
        return 0;
}
```

ii) Outside class defination

-> To define a member function ~~otor~~ outside the class defination we have to use the scope resolution :: operator along with class name and function name.

```
//code
#include<iostream>
using namespace std;
class Geeks
{
        public:
        string geekname;
        int id;
        void printname();
        void printid()
        {
                cout<<"Geek id is:" <<id;
        }
};
// outside class defination
void Geeks :: printname()
{
        cout<<"Geekname is:" <<geekname;
}
int main()
{
        Geeks obj1;
        obj1.geekname = "nyz";
        obj1.id=15;
```

```
        obj1.printname();
        cout << endl;

        obj1.printid();
        return 0;
}

//code for encapsulation
#include <iostream>
using namespace std;
class Encapsulation {
        private:
            int n;
            public:
             void set(int a)
             {
                 n = a;
             }
             int get()
             {
                 return n;
             }
};
int main()
{
        Encapsulation obj;
        obj.set(5)
        cout << obj.get();
        return 0;
}
```

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

Eg

```
class Box{
    public:
        double getVolume (void)
        {
            return length * breadth * height;
        }
    private:
        double length;
        double breadth;
        double height;
};
```

The variables length, breadth, and height are private. This means that they can be accessed only by other members of the Box class, and not by any other part of your program. This is one way encapsulation is achieved.

11. Short notes

a) Early vs late binding

=) Early binding is a phenomenon where in the decision to match various function calls happen at the compile time itself and the compiler directly associates the link with addresses. In early binding, the compiler chooses a function declaration, out of many functions with different signatures / arguments at compile time itself. The binding happens very early before the program runs.

late binding is a computer programming mechanism in which the method being called upon an object, or the function being called with arguments, is looked up by name at run time.
In case of late binding, the compiler matches the function call with the correct function definition at run time. It is also known as Dynamic Binding or Runtime Binding. late binding can be achieved by declaring a virtual function.

b) POP vs OOP

	Procedure Oriented programming	Object Oriented programming
1)	Program is divided into functions	Program is divided into objects.
2)	~~Program~~ It follows top-down approach	It follows bottom-up approach
3)	It doesn't use access specifier or modes	It uses access specifier
4)	There is no data hiding	Encapsulation is used to hide the data
5)	There is no virtual function	There is concept of virtual function
	eg:- C, Pascal etc	Eg C++, JAVA

c) Namespaces

A namespace is a declarative region that provides a scope to the identifiers (name of the types, function, variables etc) inside it. Namespace is a feature added in C++ but not in C. Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope. We can split the ~~defin~~ definition of namespace over several units. Namespace declaration don't have access specifiers. Namespace

declaration can be nested within another namespace.

```
//code
#include<iostream>
using namespace std;

namespace ns1
{
        int value()
        {
                return 5;
        }
}

namespace ns2
{
        const double x=100;
        double value()
        {
                return 2*n;
        }
}
int main()
{
        cout<<ns1::value()<<'\n';
        cout<<ns2::value()<<'\n';
        cout<<ns2::x<<'\n';
        return 0;
}
```