Assignment List.

Date

1. WAP in C++ to enter the information of 10 students. The information consists of student's id, Student's name, Course id, Course name, marks in five subjects. Also calculate the total marks and percentage. Display also the information of the student who has scored the highest mark.

```
# include <iostream.h>
# include <conio.h>
# include <stdio.h>
# define S 10
 struct student
  {
        int stdid ;
        char stdname [30];
        int cid ;
        char cname [10] ;
        int marks [5] , total ;
        float percentage ;
  };
```

Date

```
void main()
 {
    void highest_total (student s[], int n);
    student allstudents[S];
    int i, j, n;
    clrscr();
    cout << " Enter the number of students <= "<<S<<endl;
    cin >> n;
    for (i=0; i<n; i++)
     {
        clrscr();
        cout << " Enter the record of student"<<i+1
                                          <<endl;
        cout << "\n stdid : ";
        cin >> allstudents[i].stdid;
        cout << "\n stdname : ";
        gets (allstudents[i].stdname);
        cout << "\n cid : ";
        cin >> allstudents[i].cid;
        cout << "\n cname : ";
        gets (allstudents[i].cname);
```

Date

```
cout << "\n Enter marks: \n";
        for (j=0; j<5; j++)
            { cout << "\n Subject" << j+1 << ": ";
              cin >> allstudents[i].marks[j];
            }
}
clrscr();
highest_total (allstudents, n);
getch();
}
void highest_total (student s[], int n)
  {
    int i, j, position = 0, total = 0, sum;
    float percentage = 0;
    for (j=0; j<5; j++)
        total += s[0].marks[j];
    for (i=0; i<n; i++)
        { sum = 0;
          for (j=0; j<5; j++)
          sum += s[i].marks[j];
          per = sum/5;
```

Date

```
if (sum > total)
    {   total = sum;
        position = i;
    }
cout << "The total marks :" << total << endl;
cout << "The percentage is :" << percentage << endl;
}
cout << "\n Student securing highest total marks:";
cout << "\n\n student id: " << s[position].stdid << ",";
cout << "\n Student name: " << s[position].stdname << endl;
cout << "\n course id: " << s[position].cid << endl;
cout << "\n course name:" << s[position].cname << endl;
}
```

Output :

Enter the number of students <= 10

2

stdid : 1
stdname : DK
cid : 1'1
cname : BBA
subject 1 : 10
" 2 : 10
" 3 : 10
" 4 : 10
" 5 : 10

std id : 2
stdname : MG
cid : 1'3
cname : BSc
subject 1 : 20
" 2 : 20
" 3 : 20
" 4 : 20
" 5 : 20

Date

Total marks = 50
percentage = 10

Total marks = 100
percentage = 20

Student securing highest total marks:

student id : 2
Student name: MG
course id : 13
course name : BSc.

Pass by value. (Passing objects to function by value)

Date

Q. 2. WAP in C++ to add two distances using OOPs.

```
#include<iostream.h>
#include<conio.h>
class add
 {
     public : int a,b;
              void enter()
              {
                 cout << "\n Enter the values
                         of a and b : ";
                 cin >> a>>b;
              }
              void sum(add) ;

 };
void add :: sum(add ob1)
 {
          int result ;
          result = ob1.a + ob1.b ;
          cout << " result : " << result ;

};
```

Date

```
void main()
{
    one obj1, ob;
    clrscr();
    obj1.enter();
    ob.sum(obj1);
    getch();
}
```

Output :→ Enter the values of a and b :

2 4 Result : 6

A copy of the object is passed to the function and any change made is not reflected back. An object may be passed by value to a member function, a non member function and a friend function. The member function and friend functions are permitted to access the private and protected members of the passed object. The non member function can only access the public members of the passed object.

Date

Q.3. WAP in C++ to add two complex numbers using OOPS.

```
#include<iostream.h>
#include<conio.h>
class complex
{
        int real, imag ;
        public : void getdata()
                {
                    cout<<" Enter the real part: ";
                    cin >> real ;
                    cout<<" Enter the imaginary part: ";
                    cin >> imag ;
                }
                complex add (complex p)
                {
                    complex temp;
                    temp.real = real + p.real;
                    temp.imag = imag + p.imag;
                    return temp;
                }
```

Date

```
void display ()
  {
    cout << real << "+i " << imag << "\n";
  }
};
void main ()
  {
    complex a, b, c;
    a.getdata ();
    b.getdata ();
    c = a.add(b);
    cout << " first number ";
    a.display();
    cout << " second number ";
    b.display();
    cout << " The sum is ; ";
    c.display ();
    getch();
  }
```

Output: Enter the real part = 2
Enter the imagipart = 3
Enter the real part = 3
Enter the emaginary part = 4
first number 2+i3
Second number 3+i4 The sum is 5+i7.

Date

Q.4. WAP in C++ to test whether the entered date is valid date or not.

```
#include <iostream.h>
#include <conio.h>
#include <dos.h>
class date
{
    private:    int day, month, year;

    public:     date()
                {
                    struct date d;
                    getdate(&d);
                    day = d.da_day;
                    month = d.da_mon;
                    year = d.da_year;
                }

                void putdata(int m, int d, int y)
                {
                    month = m;
                    day = d;
                    year = y;
                }
                void showdata();
};
```

Date

```
void date1 :: showdata()
{
    cout << day ;
    switch (month)
    { case 1 : cout << "January ";
               break;
      case 2 : cout << " February ";
               break;
      case 3 : cout << " March ";
               break;
      case 4 : cout << "April ";
               break;
      case 5 : cout << " May ";
               break;
      case 6 : cout << " June ";
               break;
      case 7 : cout << " July ";
               break;
      case 8 : cout << " August ";
               break;
      case 9 : cout << "September ";
               break;
      case 10 : cout << " October ";
               break;
      case 11 : cout << " November ";
               break;
      case 12 : cout << " December ";
               break;
}
```

Date

```
    cout << "  " << year;
}
void main()
{   int m, d, y;
    clrscr();
    date1 d1;
    do
    {   clrscr();
        cout << "\n default date is :";
        d1.showdata();
        cout << "\n Enter the date dd mm yy:";
        cin >> m >> d >> y;
    }
    while((m<1 || m>12) || (d<1 || d>31) || (y<1 || y>2019));
    d1.putdata(m,d,y);
    cout << "\n Entered date is :";
    d1.showdata();
    if((y%4==0) && (y%100!=0)) || (y%400==0))
    {   cout << y << " is a leap year and date is
            valid \n";
    }
```

Date

```
else
{
    if (d==29 && m==2)
        cout << y << " is not a leap year and date is
            not valid \n";
}
getch();
}
```

output: →

Default date is 4 January 2011.

Enter the date mm dd yyyy.

4 4 2011

Entered date is : 4 April 2011.

Date

Q.5 WAP in C++ to show the work of copy constructor

```
#include<iostream.h>
#include<conio.h>
class fun
{
    int x, y;
    public:  fun() { }
             fun(int a, int b)
             {
                 x=a;
                 y=b;
             }
             fun(fun &f)
             {
                 cout<<"\n Copy constructor
                       is at work \n";
                 x = f.x;
                 y = f.y;
             }
             void display(void)
             { cout<<x<<" "<<y<<endl;
             }
};
```

Date

```
void main()
{
    clrscr();

    fun obj(30,70);
    fun obj2(obj);
    fun obj3(obj2);
    fun obj4;
    obj4 = obj;
    obj.display();
    obj2.display();
    obj3.display();
    obj4.display();
    getch();
}
```

output: copy constructor is at work.
copy constructor is at work.

30 70
30 70
30 70
30 70.

Date

Q.⑧. WAP in C++ to illustrate the use of this pointer if a parameter passed to a member function is the object itself.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class per
  {
      char name[20];
      float salary;
      public:   per(char *s, float a)
                {
                   strcpy(name, s);
                   salary = a;
                }
                per *gr(per &x)
                {
                   if(x.salary >= salary)
                         return &x;
                   else
                         return this;
                }
```

Date

```
void display()
{
    cout<<"name:"<<name<<"\n";
    cout<<"salary:"<<salary<<"\n";
}
};
void main()
{
    per p1("Reema", 10000), p2("Krish", 20000),
        p3("Georgi", 2000);
    per *p;
    p = p1.gr(p3);
    p->display();
    p = p2.gr(p3);
    p->display();
}
```

output:

```
name : Reema
salary : 10000
name : Krish
salary : 20000
```

Date

Q.9 WAP in C++ to use a static member for a variable within a class that can contain a counter with the number of objects of that class that are currently allocated.

```
#include<iostream.h>
#include<conio.h>
class student
{
    private : static int count;
              int rollno, marks;
    public : void enter (int r, int m)
             {
               rollno = r;
               marks = m;
               count++;
             }
             void show (void)
             {
               cout<<"\n roll number:"
                   << rollno <<"\t";
               cout << " marks :" << marks<<"\n";
             }
```

Date

```
static void showcount (void)
{
    cout << "count = " << count << "\n";
}
};
int student :: count = 0;
void main ()
{
    clrscr();
    student obj, obj2, obj3;
    obj. enter (101, 99);
    student :: showcount ();
    obj2. enter (102, 98);
    student :: showcount ();
    obj3. enter (103, 100);
    student :: showcount ();
    obj. show();
    obj2. show();
    obj3. show();
    getch();
}
```

Date

Q. 13. WAP in c++ to define a class as friend of another one, granting that first class access to the protected and private members of the second one.

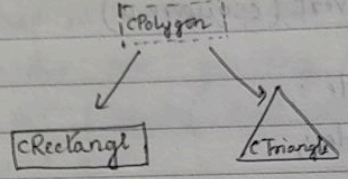
```
#include <iostream.h>
#include <conio.h>
class csquare;
class crectangle
{
    int width, height;
    public:
        int area()
        {
            return (width * height);
        }
        void convert (csquare a);
};
class csquare
{
    private:
        int side;
    public:
        void set_side (int a)
        {
            side = a;
        }
        friend class crectangle;
};
```

Date

```
void crectangle :: convert (csquare a)
{
    width = a.side;
    height = a.side;
}
int main()
{
    csquare sqr;
    crectangle rect;
    sqr.set_side(4);
    rect.convert(sqr);
    cout << rect.area();
    return 0;
}
```

Date

Q.14



The class cPolygon would contain that are common for both types of polygon. ie. width and height. And cRectangle and cTriangle would be its derived classes, with specific features that are different from one type of polygon to the other. WAP in C++ to implement the above.

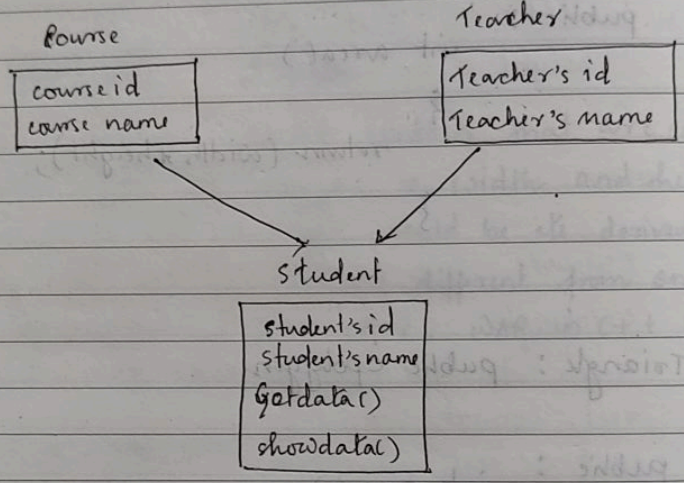
```
# include <iostream.h>
class cpolygon
  {
      protected :
                  int width , height ;
      public :
                  void set_values (int a , int b)
                  {
                     width = a ;
                     height = b ;
                  }
  };
```

Date

```
class cRectangle : public cpolygon
{
    public :
        int area()
        {
            return (width * height);
        }
};
class cTriangle : public cpolygon
{
    public : int area()
    {
        return (width * height / 2);
    }
};
int main()
{
    cRectangle rect;
    cTriangle trgl;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;
    return 0;
}
```

Date

Q.15. Implement the following with a C++ program.




```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
class course
{
    protected : char course_id[10];
                char course_name[20];
    public :    void getdata() (void);
                void showdata() (void);
};
```

Date

```
void course :: getdata1 (void)
{
    cout << " Enter the course id : \n ";
    gets (course_id);
    cout << " Enter the course name : \n ";
    gets (course_name);
}
void course :: showdata1 (void)
{
    cout << " course_id : " << course_id << endl;
    cout << " course_name : " << course_name << endl;
}
class teachers
{
    protected :    char teachers_id[10];
                   char teachers_name[20];
    public :       void getdata2 (void);
                   void showdata2 (void);
};
```

Date

```
void teacher :: getdata2()
{   cout << " Enter the teachers id: \n";
    gets (teachers_id);
    cout << " Enter the teachers name: \n";
    gets (teachers_name);
}
void teacher :: showdata2()
{   cout << " teachers id : " << teachers_id << endl;
    cout << " teachers name: " << teachers_name << endl;
}
class student : public course, public teacher
{
        char  std_id[10];
        char  std_name[20];
    public:  void getdata(void);
             void showdata(void);
};
```

Date

```
void student :: getdata()
{  cout << " Enter the required information \n";
   cout << " Enter the student id:\n";
    gets (std_id);
   cout << "Enter the student name:\n";
    gets (std_name);
     getdata1();
     getdata2();
}
void student :: showdata ()
{
    cout << " The information are:\n";
    cout << " std_id :" << std_id << endl;
    cout << " std_name:" << std_name << endl;
     showdata1();
     showdata2();
}
void main()
{    student ob;
       clrscr();
       ob.getdata();
       ob.showdata();
       getch();
}
```

Date

Q.16. " If the base class has no default constructor or you want that an overloaded constructor is called when a new derived object is created, you can specify it in each constructor definition of the derived class." WAP in c++ in support of the above statement.

```
// constructors and derived classes.
# include <iostream.h>
class mother
{
    public :  mother()
              {
                cout << "mother : no parameter\n";
              }
              mother (int a)
              {
                cout << " mother : int parameter\n";
              }
};
```

Date

```
class daughter : public mother
{
    public :  daughter (int a) // nothing specified: call default
              {
                cout << "daughter : int parameter\n\n";
              }
};

class son : public mother
{
    public :  son (int a) : mother (a) // constructor specified: call this
              {
                cout << " son : int parameter\n\n";
              }
};

int main ()
{
    daughter pinky (0);
    son      raj (0);
    return 0;
}
```

Date

output : mother : no parameter
daughter : int parameter
mother : int parameter
son : int parameter.

9. (13). WAP in C++ where an abstract class is inherited to a derived class.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>   // for strcpy()
const LEN = 31;
class employee
{
    private :   int employee_number ;
                char name[LEN];
                char designation[LEN];
                float basic-pay;

    public :    employee();   // default constructor
                              prototype.
```

Date

```
employee ( int empno, char tname[LEN], char desig[LEN],
                float bpay)
    {
        employee_number = empno;
        strcpy (name, tname);
        strcpy (designation, desig);
        basic_pay = bpay;
    }
void display (void)
    {
        cout << "Employee number : " << employee_number;
        cout << "\n\n Name : " << name;
        cout << "\n\n Designation : " << designation;
        cout << "\n\n Basic pay : " << basic_pay;
    }
};
employee :: employee()
    {
        employee_number = 0;
        for ( int i = 0; i < LEN ; i++)
            name[i] = ' ';
        for (i=0 ; i<LEN; i++)
            designation[i] = ' ';
        basic_pay = 0.0;
    }
```

Date

```
class executive : public employee
  {  private :  float conveyance ;
     public :   executive (float convey , int empno ,
                  char tname [LEN] , char desig [LEN] ,
                  float bpay ) : employee (empno ,
                                  tname , desig , bpay)
                {
                   conveyance = convey ;
                }
                void display (void)
                {  employee :: display ( ) ;
                   cout << "\n\n Conveyance : " <<
                              conveyance ;
                }
  } ;
void main ( )
  {
     clrscr ( ) ;
     executive obj ( 5000.0 , 1000 , 'Raman Sharma' , 'System Analyst' ,
                                  30000.0) ;
     cout << "\n\n                output . . . . \n\n" ;
  }  obj. display ( ) ;
```

Date

output :→

Employee number : 1001
Name : Raman Sharma.
Designation : System Analyst
Basic pay : 30000
Conveyance : 5000.

Q.18. WAP. in C++ to solve the problem of data ambiguity arises in hybrid inheritance.

```
// here two copies of ONE inherited to FOUR.
#include <iostream.h>
#include <conio.h>
class ONE
{
    public : int a;
};
class TWO : public ONE
{
    public : int b;
};
```

Date

```
class THREE : public ONE
  { public :   int c ;
  };
class FOUR : public TWO, public THREE
  {   public :  int d ;
  };
void main()
  {
    FOUR obj ;
    clrscr();
    // no ambiguity now.
    obj. TWO :: a = 100 ;   // a inherited from TWO
    obj. THREE :: a = 200 ;  // a  "      "  THREE.
    obj. c = 300
    obj. d = obj. TWO :: a + obj. THREE :: a + obj. c

    cout << "  ....  OUTPUT ....  \n\n ";
    cout << " a of class TWO = " << obj. TWO :: a << "\n";
    cout << " a of class THREE = " << obj. THREE :: a << "\n";
    cout << " c of class THREE = " << obj. c << "\n";
    cout << " d of class FOUR = " << obj. d ;
  }
```

Date

OUTPUT

a of class TWO = 100
a of class THREE = 200
c of class THREE = 300
d of class FOUR = 600.

9.6 WAP in C++ to illustrate the work of constructor overloading.

```
# include <iostream.h>
class CRectangle
  {    int width, height;
       public:  CRectangle();  // default constructor
                CRectangle (int, int);  // overloaded
                int area()
                { return (width * height);
                }
  };
```

Date

```
cRectangle :: cRectangle ()
  { width = 5;
    height = 5;
  }

cRectangle :: cRectangle (int a, int b)
  { width = a;
    height = b;
  }
int main ()
  {
    cRectangle  rect (3,4) ;
    cRectangle  rectb ;
    cout << "rect area :" << rect. area () << endl;
    cout << "rectb area; " << rectb. area () << endl;
    return 0;
  }
```

output :→ rect area : 12
rectb area : 25.

Date

* In this case, rectb was declared without any arguments, so it has been initialized with the constructor that has no parameters, which initializes both width and height with a value of 5.

** if we declare a new object and we want to use its default constructor (the one without parameters), we do not include parentheses ():

```
cRectangle rectb;   // right
cRectangle rectb(); // wrong.
```

Date

Q⑦. WAP in C++ to illustrate the work of operator overloading.

```
// overloading unary operator '++'
#include<iostream.h>
#include<conio.h>
class counter
{
    private: int count;
    public:
        counter()          // default constructor
        { count = 0;
        }
        counter(int c)     // constructor with one argument.
        { count = c;
        }
        int getvalue()
        { return (count);
        }
        counter operator ++ ()   // overloading unary operator.
        {
            return (counter(++count));
        }
};
```

Date

```
void main()
{ clrscr();
  counter c1, c2(10);   // objects declared
  cout << "Default values: \n c1 = " << c1.getvalue()
       << "\n c2 = " << c2.getvalue();

  ++c1;    // unary operator applied to an object
  cout << "\n The value of c1 after incrementing ="
       << c1.getvalue();
  ++c2;    // unary operator applied to an object
  cout << "\n The value of c2 after incrementing ="
       << c2.getvalue();

  c1 = ++c2;
  cout << "\n The value of c1 after incrementing c2
       and assigning to c1 = " << c2.getvalue();

  getch();
}
```

Date

output :

Default values : C1 = 0

C2 = 10

The value of C1 after incrementing = 1

The " " C2 " " = 11

The " " C1 " " " C2 and assigning to C1 = 12.

Date

```
#include <iostream.h>
class cvector
{
  public: int x, y;
          cvector () { } ;
          cvector ( int ,int ) ; // constructor
          cvector Operator+ (cvector) ;
                          // function returns
                             a cvector.
};
cvector :: cvector (int a ,int b)
   {
       x=a ;
       y=b ;
   }
cvector cvector :: operator + (cvector param)
   {
      cvector temp ;
      temp.x = x + param.x ;
      temp.y = y+ param.y ;
      return (temp) ;
   }
int main()
   { cvector a (3,1) ;
     cvector b (1,2) ;
     cvector c ;
     c = a+b ;
     cout <<c.x <<" ," <<c.y ;
   return 0;
```

Date

output : 4,3.

The function operator(+). This function can be called either implicitly using the operator, or explicitly using the function name:

```
c=a+b.
c=a.operator+(b);
```
} equivalent.

Date

Polymorphism in C++.

The word polymorphism means having many forms.
i.e we can define polymorphism as the ability of a message to be displayed in more than one form.

Real life example of polymorphism, a person at the same time can have different characteristic.
like a man at the same time is a father, a husband, an employee.
So the same person posses different behavior in different situations.
this is called polymorphism.
Polymorphism is considered as one of the important features of OOP.

In C++ polymorphism is mainly divided into two types:

1. Compile time polymorphism (static)
2. Runtime Polymorphism. (Dynamic).