1.Write a C++ program to create a class named as Employee with private attributes emp_id and emp_salary. Provide appropriate public interfaces to initialize these attributes and display the details.

```
#include<iostream>
using namespace std;
class employee
{
private:
    int emp_id;
    double emp_salary;
public:
    void show()
    {
        cout<<"Salary of id "<<emp_id<<" is "<<emp_salary<<endl;
    }
    void getdata()
    {
        cout<<"Enter the employee id = ";
        cin>>emp_id;
        cout<<"Enter the employee salary = ";
        cin>>emp_salary;
    }
};
int main()
{
    employee e1;
    e1.getdata();
    e1.show();
    return 0;
}
```

2.Write a C++ program to create a class named as Triangle with three sides as its attributes. The attributes need to be placed under protected visibility. Provide a function to initialize these attributes.
Provide another method which finds out the area of a triangle object and displays the value of the area at the console.

```
#include<iostream>
#include<math.h>
using namespace std;

class Triangle
{
protected:
    double a,b,c;
public:
    void getdata()
    {
        cout<<"Enter the first side of a triangle = "<<endl;
        cin>>a;
        cout<<"Enter the second side of a triangle = "<<endl;
        cin>>b;
        cout<<"Enter the third side of a triangle = "<<endl;
        cin>>c;
    }
    double area()
    {
        double s;
        s=(a+b+c)/2;
        return sqrt(s*(s-a)*(s-b)*(s-c));
    }
};
int main()
{
    Triangle t;
    t.getdata();
    t.area();
    cout<<"Area of the triangle is "<<t.area();
    return 0;
}
```

3.Write a C++ program to create a class named as Customers which would store the details about a bank customer. The private attributes of the class are account_number, balance_amount and PAN_number.
Provide appropriate public interfaces to initialize these attributes and display at the console. At least 25 different customers' details need to be shown at the console.

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

class Customer

{
  private: int account_number;
   long int balance_amount, PAN_number;
  public:
       void get()
        { cout<<" ENTER THE ACCOUNT NUMBER";
         cin>>account_number;
         cout<<"ENTER THE BALANCE AMOUNT";
         cin>>balance_amount;
         cout<<"ENTER THE PAN DETAILS";
         cin>>PAN_number;
         }
```

```
      void display()
        { cout<<" THE ACCOUNT NUMBER is "<<account_number<<endl;
         cout<<" THE BALANCE AMOUNT"<<balance_amount<<endl;
         cout<<" THE PAN DETAILS"<<PAN_number<<endl;
         }




        };

        int main()
        { Customer a;
          int y;
          while(y!=0)
            {a.get();
             a.display();
            cout<<" ENTER ANOTHER DETAILS 1 if yes 0 if no";
             cin>>y;
           }
                return 0;
         }
```

4.Write a C++ program to create a class named as Convert which contains a private attribute named as seconds. Provide a public interface that would accept a positive integer value from a user and initialize the attribute seconds. Provide another public interface that would convert the seconds value into hours, minutes and seconds and displays in the following format: Hours:Minutes:Seconds.

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

class Convert

{
  private: int seconds;
  public:
       void input()
        { cout<<"ENTER THE TIME IN TOTAL SECONDS = ";
         cin>>seconds;
         }

         void CLOCK()
         {
           int h,m,s;
            h=seconds/3600;
            m=((seconds%3600)/60);
```

```
          s=((seconds%3600)%60);
         cout<<"THE TIME IN HRS:MIN:SEC IS "<<h<<":"<<m<<":"<<s; }
      };

      int main()
      { Convert a;

      a.input();
        a.CLOCK();
       return 0;
       }
```

5.Write a C++ program to create a class named as Shape with private attributes x and y. Provide a public interface named as area() which would be overloaded to initialize these attributes and find out area of a square object or of a rectangle object and displays the value at the console.

```
#include<iostream>
using namespace std;
class Shape
{
private:
    double x,y;
public:
    double area(double a)
    {
        x=a;
        return x*x;
    }
    double area(double a,double b)
    {
        x=a;
        y=b;
        return x*y;
    }

};
int main()
{
    Shape s;
    double a,b;
    cout<<"Enter the side of a square = ";
    cin>>a;
    cout<<"The area of the square is "<<s.area(a)<<endl;
    cout<<"Enter the sides of a rectangle = ";
    cin>>a>>b;
    cout<<"The area of the rectangle is "<<s.area(a,b)<<endl;
    return 0;
}
```

6.Write a C++ program to create a class named as Triangle with three sides as its protected attributes. Provide a constructor to initialize these attributes.
 Provide a public member function named as check_and_display() which would check whether the triangle is a right angled triangle or not and makes a call to another member function named as area() which would be overloaded to find out the area of a right angled triangle or any other triangle and displays the value.

```
#include<iostream>
#include<math.h>
using namespace std;
class Triangle
{
protected:
    double s1,s2,s3;
public:
    Triangle(double a, double b, double c)
    {
        s1 = a;
        s2 = b;
        s3 = c;
    }
    void area(double  base,double height)
    {
    double ar = 0.5*base*height;
    cout<<"Area of right angle triangle = "<<ar<<endl;
    }
    void area(double a, double b, double c)
    {

    double s = (a+b+c)/2;
    double z = s*(s-a)*(s-b)*(s-c);
    double ar = pow(z,0.5);
    cout<<"Area of triangle = "<<ar<<endl;
    }
    void check_and_display()
    {
        double p,q,r,m;
        p = s1;
        q = s2;
        r = s3;
        if(p>q&&p>r)
        {
            m = p;
            if(m*m == q*q + r*r)
```

```
                area(q,r);
            else
                area(p,q,r);
        }
        else if(q>p&&q>r)
        {
            m = q;
            if(m*m == p*p + r*r)
                area(p,r);
            else
                area(p,q,r);
        }
        else if(r>p&&r>q)
        {
            m = r;
            if(m*m == p*p + q*q)
                area(p,q);
            else
                area(p,q,r);
        }
    }
};
int main()
{
    double p,q,r;
    cout<<"Enter side 1 "<<endl;
    cin>>p;
    cout<<"Enter side 2 "<<endl;
    cin>>q;
    cout<<"Enter side 3 "<<endl;
    cin>>r;
    Triangle t(p,q,r);
    t.check_and_display();
    return 0;
}
```

7.Write a complete C++ program to count the number of objects created for a class. Use static member variable and constructor.

```
#include<iostream>
using namespace std;
class A
{
public:
    int x;
    static int count;
    count=0;

    A()
    {
        count++;
        cout<<count<<" object created"<<endl;
    }
};

int main()
{
    A a,b,c;
    cout<<"Total no. of objects created = "<<A::count<<endl;
    return 0;
}
```

8.With a suitable example explain the ambiguity arise while overloading a function with default parameters.

```
#include<iostream>
using namespace std;
void fun(int& x,int y =10)
{
    int s = x+y;
    cout<<"sum = "<<s;
}
void fun(int& a)
{
    int s = a;
}
int main()
{
    int c,d;
    cin>>c;
    cin>>d;
    fun(c,d);
    fun(c);
    return 0;
}
```

9.Create a base class with a constructor having single int type parameter. Create another base class with a constructor having three int type parameters. Now create a derived class from the above two base classes.
The derived class constructor should be able to pass appropriate number of parameters to the base class constructors.

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

```
class Base1
{
public:
    int x;
    Base1(int i)
    {
        x=i;
    }
};
class Base2
{
public:
    int p,q,r;
    Base2(int a,int b,int c)
    {
        p=a;
        q=b;
        r=c;
    }
};
class Deri : public Base1,public Base2
{
public:
    Deri(int g,int h,int j):Base1(g),Base2(g,h,j)
    {
        x=g;
        p=g;
        q=h;
        r=j;
    }
    void show()
    {
        cout<<"x = "<<x<<endl;
        cout<<"p = "<<p<<" ,q = "<<q<<" ,r = "<<r<<endl;
    }
};
int main()
{
    Deri ob1(4,6,9);
    ob1.show();
    return 0;
}
```

10.How ambiguity can be resolved for multiple copies of a same attribute in case of multipath(diamond) inheritance using virtual base class? Explain with an example.

```
#include <iostream>
using namespace std;
class ClassA
{
public:
    int a;
};

class ClassB : virtual public ClassA
{
public:
    int b;
};
class ClassC : virtual public ClassA
{
public:
    int c;
};

class ClassD : public ClassB, public ClassC
{
public:
    int d;
};

int main()
{
    ClassD obj;
    obj.a = 30;
    obj.b = 35;
    obj.c = 60;
    obj.d = 72;
    cout<< "\n A : "<< obj.a;
    cout<< "\n B : "<< obj.b;
    cout<< "\n C : "<< obj.c;
    cout<< "\n D : "<< obj.d;
    return 0;

}
```

11.An educational organization maintains the below given structure for smooth functioning. Provide necessary methods to initialize these attributes and display the details.

In the below given diagram all the classes and their respective attributes are given.

All the attributes are under protected visibility and inheritance is done in public mode. The Gross should be calculated as follows: Gross= Basic_pay + DA + HRA, where DA= 1.5 * Basic_pay, HRA= 0.2 * Basic_pay.

```
#include <iostream>

using namespace std;
class Regular
{
protected:
    double Basic_pay, DA, HRA, Gross;
public:
    Regular()
    {
        cout<<"Enter Basic pay: ";
        cin>>Basic_pay;
        DA=1.5*Basic_pay;
        HRA=0.2*Basic_pay;
        Gross=Basic_pay+DA+HRA;
    }

};
class Contract
{
protected:
    double Pay_consolidated;int Contract_duration_months;
public:
    Contract()
    {
        cout<<"Enter Pay consolidated: ";
        cin>>Pay_consolidated;
        cout<<"Enter Contract duration months: ";
        cin>>Contract_duration_months;
    }
};
class Teacher:public Regular,public Contract
{
protected:
    int years_of_experience;
public:
    Teacher()
    {
        cout<<"Number of years: ";
        cin>>years_of_experience;
    }
};
class Officer
{
protected:
    char grade;
public:
    Officer()
    {
        cout<<"Grade: ";
        cin>>grade;
    }
};
class Staff:public Teacher,public Officer
{
protected:
    string id,phone;
public:
    Staff()
    {
        cout<<"ID: ";
        cin>>id;
        cout<<"Phone: ";
        cin>>phone;
    }
    void display()
    {
        cout<<"\nThe details are:\n";
        cout<<"Basic pay: "<<Basic_pay<<endl;
        cout<<"DA: "<<DA<<endl;
        cout<<"HRA: "<<HRA<<endl;
        cout<<"Gross: "<<Gross<<endl;
        cout<<"Pay consolidated: "<<Pay_consolidated<<endl;
        cout<<"Contract duration months: "<<Contract_duration_months<<endl;
        cout<<"Number of years: "<<years_of_experience<<endl;
        cout<<"Grade: "<<grade<<endl;
        cout<<"ID: "<<id<<endl;
        cout<<"Phone: "<<phone<<endl;
    }
};
int main()
{
    Staff s;
    s.display();
}
```

12.Develop a program to prevent inheritance. For example, suppose you have a class called base. You have to develop a program so that, no further derived classes can be created from that base class.

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

```
class Base1{
    private:
        Base1(){
            cout<<"In base 1"<<endl;
        }

};

class Derived1:Base1{

};

int main(){
    Derived1 ob;
    return 0;
}
```

13.Create a class called Figure which contains an attribute of type double that could be used to compute the area of figures. Derive two specific classes called Circle and Square from the base Figure.
Add to the base class a pure virtual member function show_area() to compute and display the area of figures. The show_area() function need to be redefined in the respective derived classes. Provide constructors to initialize the attributes.

```
#include <iostream>
using namespace std ;
class Figure
{
protected :
    double x ;
public:
    Figure (double a)
    {
        x = a ;
    }
virtual void show_area()=0;
} ;
class Circle: public Figure
{
public :
    Circle (double b) : Figure(b) { } ;
    void show_area()
    {
        cout<<"area = "<<x*x*3.14<<endl ;
    }
};
class Square : public Figure
```

```
{
public:
    Square (double b) : Figure(b) { } ;
    void show_area()
    {
        cout<<"area = "<<x*x<<endl ;
    }
} ;
int main()
{
Square a(5) ;
a.show_area();
Circle b(10) ;
b.show_area();
return 0 ;
}
```

14. Write a complete program to create a class named as Container which contains an attribute of type double and a public pure virtual function named as volume(). Create two new classes named as sphere and cube from the above class. Implement dynamic polymorphism to find the volumes of a sphere and cube objects by redefining the volume() function.
You can provide any other function to initialize the attribute and to display the volume.

```
#include<iostream>
using namespace std;
const double PI=3.14;

class Container
{
   protected:
   double x;
   public:
         virtual void init()=0;
         virtual void volume()=0;
};

class Cube:public Container
{
   public:
         void init()
  {
   cout<<"Enter length of the side of the cube"<<endl;
   cin>>x;
  }
         void volume()
         {
            cout<<"Volume of the cube is="<<x*x*x<<endl;
         }
};

class Sphere:public Container
{
   public:
         void init()
 {
   cout<<"Enter length of the radius of the sphere"<<endl;
   cin>>x;
 }
         void volume()
         {
            cout<<"Volume of the sphere is="<<((double)4/3)*PI*x*x*x<<endl;
         }
};

int main()
{
   Container *c;
   Cube cb;
   c=&cb;
   c->init();
   c->volume();
   Sphere s;
   c=&s;
   c->init();
   c->volume();
   return 0;
}
```

15.With an appropriate example explain how dynamic polymorphism can be achieved using a base type reference variable?

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

class Base
{
   public:
         int x;
 virtual void show()
 {
     x=100;cout<<"x in base = "<<x<<endl;
         }
};

class Derived:public Base
{
   public:
         int y;
 void show()
 {
    x=200;
y=400;
cout<<"x in derived = "<<x<<endl;
cout<<"y in derived = "<<y<<endl;
 }
};
void display(Base &ref)
{
   ref.show();
}

int main()
{
   Base b;
   display(b);
   Derived d;
   display(d);
   return 0;
}
```

16. With an appropriate example show that virtual functions are inherited in multi-level inheritance.

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

class Base
{
   public:
         int x;
 virtual void show(){x=0;};
};

class Derived1:public Base
{
   public:
         int y;
 void show()
 {
    x=33;
y=40;
cout<<"x in derived = "<<x<<endl;
cout<<"y in derived = "<<y<<endl;
cout<<"\n";
 }
};

class Derived2:public Derived1
{
   public:
         int y;
 void show()
 {
    x=22;
y=45;
cout<<"x in derived = "<<x<<endl;
cout<<"y in derived = "<<y<<endl;
cout<<"\n";
 }
};

int main()
{
   Base *bptr;
   Derived2 d;
   bptr=&d;
   bptr->show();
   return 0;
}
```

17.Develop a C++ program to perform addition, subtraction, multiplication and division of two user defined type variables. The user defined type variables contain two protected real numbers respectively.
The binary +, -, * and / operators should be overloaded.

```
#include<iostream>
#include<math.h>
using namespace std;
class cal
{
protected:
    double x;
    double y;
public:
    cal(){}
    cal(int p, int q)
    {
        x = p;
        y = q;
    }
    cal operator+(cal t)
    {
        cal temp;
        temp.x = x + t.x;
        temp.y = y + t.y;
        return temp;
    }
    cal operator-(cal t)
    {
        cal temp;
        temp.x = x - t.x;
        temp.y = y - t.y;
        return temp;
    }
    cal operator*(cal t)
    {
        cal temp;
        temp.x = x * t.x;
        temp.y = y * t.y;
        return temp;
    }
    cal operator/(cal t)
    {
        cal temp;
        temp.x = x / t.x;
        temp.y = y / t.y;
        return temp;
    }
    void display()
    {
        cout<<"x = "<<x<<endl;
        cout<<"y = "<<y<<endl;
    }
};
int main()
{
    cal a(8,4),b(2,4),c;
    c = a+b;
    cout<<"a + b : "<<endl;
    c.display();
    c = a-b;
    cout<<"a - b : "<<endl;
    c.display();
    c = a*b;
    cout<<"a * b : "<<endl;
    c.display();
    c = a/b;
    cout<<"a / b : "<<endl;
    c.display();
    return 0;
}
```

18.Create a class named as Test with two integer attributes x and y. Provide an operator function to check whether two objects of the class Test are equal or not.
The comparison operator (= =) should be overloaded.

```
#include<iostream>
#include<math.h>
using namespace std;
class Test
{
protected:
    int x;
    int y;
public:
    Test(){}
    Test(int p, int q)
    {
        x = p;
        y = q;
    }
    bool operator==(Test t)
    {
        if(x==t.x&&y==t.y)
            return true;
        else
            return false;
    }
};
int main()
{
    Test t1(3,4),t2(3,4),t3(4,5);
    if(t1==t2)
    {
        cout<<"t1 and t2 are equal"<<endl;
    }
    else
    {
        cout<<"t1 and t2 are not equal"<<endl;
    }
    if(t1==t3)
    {
        cout<<"t1 and t3 are equal"<<endl;
    }
    else
    {
        cout<<"t1 and t3 are not equal"<<endl;
    }
    return 0;
}
```

19.Develop a C++ program where a user defined type variable can be subtracted from an integer value and vice versa. The user defined type variable should contain only a single integer attribute under protected visibility.

```
#include<iostream>
#include<math.h>
using namespace std;
class A
{
protected:
    int x;
public:
    A()
    {
        cout<<"Enter x"<<endl;
        cin>>x;
    }
    friend int operator-(int y,A ob)
    {
        int dif2;
        dif2 = y-ob.x;
        return dif2;
    }
    friend int operator-(A ob,int y)
    {
        int dif;
        dif = ob.x-y;
        return dif;
    }
};
int main()
{
    int y;
    cout<<"Enter an integer"<<endl;
    cin>>y;
    A a;
    int d1,d2;
    d1 = a-y;
    cout<<"a-y = "<<d1<<endl;
    d2 = y-a;
    cout<<"y-a = "<<d2<<endl;
    return 0;
}
```

20.C++ does not provide any mechanism to check array boundary overrun or under run.
Create a class where an array indexing operator can be overloaded to check the boundary conditions for an array which is an attribute of that class.

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

class Point
{
   int arr[3];
   public:
        Point(){}
Point(int a,int b,int c)
{
   arr[0]=a;
   arr[1]=b;
   arr[2]=c;
}
int& operator[](int k)
{
  if(k<0||k>2)
  {
    cout<<"trying to access beyond the array boundary"<<endl;
exit(1);
  }
  else
    return arr[k];
}
};

int main()
{
   Point p(10,20,30);
   cout<<"First element="<<p[0]<<endl;
   cout<<"Second element="<<p[1]<<endl;
   cout<<"Third element="<<p[2]<<endl;
   p[0]=100;
   cout<<"The new value of p[0]="<<p[0]<<endl;
   p[3]=300;
   return 0;
}
```

21. Write a C++ program to allocate memory for an array of double type. The size of the array should be given by the user at the run time.
Use new and delete operators for memory allocation and deallocation.

```
#include<iostream>
using namespace std;
int main()
{
    double *p;
    int n;
    cout<<"Enter the size of the array"<<endl;
    cin>>n;
    p = new double[n];
    for(int i=0;i<n;i++)
    {
        cout<<"Enter a number"<<endl;
        cin>>p[i];
    }
    cout<<endl;
    for(int j=0;j<n;j++)
    {
        cout<<"The number is : "<<p[j]<<endl;
    }
    delete []p;
    return 0;
}
```

22. Create a class named as student with protected attributes such as stud_id and stud_mark. Take a constructor to initialize these attributes.
Create an array of objects for the above class and initialize the attributes. The memory allocation for that array should take place at the run time. Provide appropriate method to display the initialized values.

```
#include<iostream>
using namespace std;
class student
{
protected:
    int stud_id;
    double stud_mark;
public:
    student()
    {
        cout<<"Enter student id:"<<endl;
        cin>>stud_id;
        cout<<"Enter student mark:"<<endl;
        cin>>stud_mark;
    }
    void display()
    {

        cout<<"Student details "<<" : "<<endl;
        cout<<"ID : "<<stud_id<<endl;
        cout<<"Mark : "<<stud_mark<<endl;
    }
};
int main()
{
    student *s;
    cout<<"Enter the no. of students"<<endl;
    int n;
    cin>>n;
    s = new student[n];
    cout<<endl;
    for(int i=0;i<n;i++)
    {
        s[i].display();
    }
    delete []s;
    return 0;
}
```

23.With an appropriate example explain, what is the role of a copy constructor when a class contains a pointer attribute and memory allocation for such an attribute takes place at the run time?

```
#include<iostream>
using namespace std;
class person
{
    int *age;
public:
    person(int a)
    {
        age = new int;
        *age = a;
    }
    void show()
    {
        cout<<"Age = "<<*age<<endl;
    }
    ~person()
    {
        delete age;
    }
    /*person(const person&copy)//shallow copy
    {
        age = copy.age;
    }*/
    person(const person&copy)//deep copy
    {
        age = new int;
        *age = *(copy.age);
    }
};
int main()
{
    person p1(18);
    p1.show();
    {
        person p2=p1;
        p2.show();
    }//destructor called for p2
    p1.show();
    return 0;
}//destructor called for p1
```

24.Create a linked list with 5 nodes. Each node contains an integer data field and a pointer to the next node. After inserting data into the nodes sort the nodes in ascending order of their respective data values and display the sorted list.
Memory management for the nodes should be done dynamically with the help of new and delete operators.

```
#include<iostream>
using namespace std;
class A
{
public:
    int d;
    A *ptr;
};
int main()
{
    A *a;
    a = new A[5];
    int i;
    for(i=0;i<4;i++)
    {
        cout<<"Enter data for node: "<<i+1<<endl;
        cin>>a[i].d;
        a[i].ptr = &a[i+1];
    }
    cout<<"Enter data for node: 5"<<endl;
    cin>>a[i].d;
    a[i].ptr = NULL;
    int j;
    for(j=0;j<4;j++)
    {
        A temp;
        temp.ptr = &a[j];
        if(a[j].d>a[j+1].d)
        {
            temp.d = a[j].d;
            a[j].d = a[j+1].d;
            a[j+1].d = temp.d;
        }
        a[j].ptr = a[j+1].ptr;
        a[j+1].ptr = temp.ptr;
    }
    cout<<endl;
    A *t;
    t = &a[0];
    for(int q=0;q<5;q++)
    {
        cout<<a[q].d<<endl;
    }
    delete []a;
    return 0;
}
```

25. Write a program where a single catch block is capable of handling any type of exception.

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

void func(int a)
{
   cout << "Throwing exception\n";
   if(a==0)
```

```
      throw 404;
   else if(a==1)
      throw 79.8;
   else if(a==2)
      throw 'Z';
   else
      throw "Wow";

}
int main()
{
   cout<<"Enter a number: "<<endl;
   int num;
   cin >> num;
   try
   {
      func(num);
   }
   catch(...){cout<<"Caught an exception"<<endl;}

   return 0;
}
```

26.Write a program where a user defined function is capable of throwing exceptions of type double only.

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

void func(int a) throw(double)
{
   cout << "Throwing exception\n";
   if(a==0)
      throw 404;
   else if(a==1)
      throw 79.8;
   else if(a==2)
      throw 'Z';
   else
      throw "Wow";

}
int main()
{
   cout<<"Enter a number: "<<endl;
   int num;
   cin >> num;
```

```
  try
  {
    func(num);
  }
  catch(...){cout<<"Caught an exception"<<endl;}

  return 0;
}
```

27. With a suitable example explain how ordering of catch blocks plays an important role while dealing with exceptions of both base type and derived type.

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

class Base
{};
class Derived:public Base
{};

int main()
{
  Base bobj;
  Derived dobj;
  try
  {
    cout<<"Inside try"<<endl;
 throw dobj;
  }

  catch(Derived tempd)
  {
    cout<<"Caught a derived"<<endl;
  }
  catch(Base tempb)
  {
    cout<<"Caught a base"<<endl;
  }

  cout<<"End"<<endl;

  return 0;
}
```

28. Write a program where an exception of type float can be rethrown.

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

void func(){
    try{
        cout << "In try of func()\n";
        throw 4.4f;
    }
    catch(float e){
        cout << "Caught exception: " << e << "\n";
        throw;
    }
}

int main() {
    try{
        cout << "In try of main()\n";
        func();
    }
    catch(float be) {
        cout << "Caught rethrown exception: " << be << "\n";
    }

    return 0;
}
```

29.Write a program that contains a generic function that can swap any two values.

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

template<class T>
void swapAny(T &a, T &b)
{
   T temp;
   temp=a;
   a=b;
   b=temp;
}

int main()
{
   int i1 = 10, i2 = 20;
   double d1 = 45.1, d2 = 33.6;
   char c1 = 'A', c2 = 'Z';
   cout << "\ni1=" << i1 << " i2=" << i2 << endl;
   cout << "d1=" << d1 << " d2=" << d2 << endl;
   cout << "c1=" << c1 << " c2=" << c2 << endl;

   swapAny(i1,i2);
   swapAny(d1,d2);
   swapAny(c1,c2);

   cout << "\nResult :\n";
   cout<<"i1="<<i1<<" i2="<<i2<<endl;
   cout<<"d1="<<d1<<" d2="<<d2<<endl;
   cout<<"c1="<<c1<<" c2="<<c2<<endl;

   return 0;
}
```

30.Write a program which contains a generic function that can arrange elements of an array in ascending order.

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

template<class T>
void sortAny(T a[], int n) {
    int min = 0;
    for (int i = 0 ; i < n-1 ; ++i) {
        min = i;
        for (int j = i ; j < n ; ++j) {
            if (a[min] > a[j])
                min = j;
        }
        if (min != i) {
            int temp = a[i];
            a[i] = a[min];
            a[min] = temp;
        }
    }
}

int main() {
    int size;
    cout << "Enter array size: ";
    cin >> size;
    int arr[size];

    cout << "Enter array elements: ";
    for (int i = 0 ; i < size ; ++i)
```

```
        cin >> arr[i];

    sortAny(arr, size);

    cout << "Sorted array: \n";
    for (int i = 0 ; i < size ; ++i)
        cout << arr[i] << " ";
    cout << "\n";

    return 0;
}
```

31. Write a program to overload a generic function along with specific version of the same function.

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

template<class A, class B> void func(A a, B b)
{
   cout << "First function called: \n";
   cout << "A = " << a << " B = " << b << endl;
}

template<class A> void func(A a)
{
   cout << "Second function called: \n";
   cout << "A = " << a << endl;
}

int main()
{
   func(35);
   func(98.4, "Wow");

   return 0;
}
```

32. Write a program to create a generic class named as Stack to simulate the behavior of a stack. The stack should contain push() and pop() functions to insert and delete elements respectively.
The stack may contain a display() function to display the top element of a stack.

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

template<class T> class Stack
```

```
{
   T tarr[5];
   int top;
   public:
         void init()
 {
    top=-1;
 }

         void push(T temp)
 {
    if(top>=4)
               cout<<"Stack Overflow"<<endl;
            else {
                  top++;
                  tarr[top]=temp;
         }
 }

 void pop()
 {
    if(top<=-1)
               cout<<"Stack Underflow"<<endl;
            else {
                  cout<<"The popped element is "<<tarr[top]<<endl;
                  top--;
                }
 }

 void show()
 {
            if(top>=0)
{
               cout<<"The top element of stack is ";
                   cout<<tarr[0];
            }
else
               cout<<"Stack is empty";
            }
};

int main()
{
   Stack<double> stack1;
   stack1.init();
   for(int k=0;k<5;k++)
```

```
  {
  double dd;
  cout<<"Push doubles onto the stack"<<endl;
  cin>>dd;
      stack1.push(dd);
  }
  stack1.pop();
  stack1.show();
  return 0;
}
```

33.Create two different header files. Each of them contains a function to sort integers. The functions signature remains same in both the cases.
In the first case the sort function sorts the elements in ascending order whereas in second case the sort function sorts the elements in descending order.
Create another c++ file where these two header files are included and the sort functions are called to perform the appropriate operation. There should not be any ambiguity while calling these functions.

```
//code of the header files are below.
#include "asc.h"
#include "des.h"
#include<iostream>
using namespace std;

int main()
{
    int n;
    cout<<"Size: "<<endl;
    cin>>n;
    int arr[n];
    for(int i = 0 ; i < n ; i++)
        cin>>arr[i];

    asc::srt(arr, n);
    cout<<"Ascending array: "<<endl;
    for(int i = 0 ; i < n ; i++)
        cout<<arr[i]<<endl;

    des::srt(arr, n);
    cout<<"Descending array: "<<endl;
    for(int i = 0 ; i < n ; i++)
        cout<<arr[i]<<endl;

    return 0;
}
```

```
asc.h
namespace asc{
    int srt(int Arr[], int n)
    {
        for(int i = 1 ; i < n ; i++)
        {
            for(int j = 0 ; j < n - i ; j++)
            {
                if(Arr[j] > Arr[j+1])
                {
                    int temp = Arr[j];
                    Arr[j] = Arr[j+1];
                    Arr[j+1] = temp;
                }
            }
        }

        return Arr[n];
    }
}
```

```
des.h
namespace des{
    int srt(int Arr[], int n)
    {
        for(int i = 1 ; i < n ; i++)
        {
            for(int j = 0 ; j < n - i ; j++)
            {
                if(Arr[j] < Arr[j+1])
                {
                    int temp = Arr[j];
                    Arr[j] = Arr[j+1];
                    Arr[j+1] = temp;
                }
            }
        }

        return Arr[n];
    }
}
```

34. Create a class named as employee with protected attributes such as emp_id and emp_salary. Provide constructor to initialize these members. Provide appropriate member method to display the initialized values.
The class should be declared and defined within a user defined namespace. Create an object of the class within the user defined namespace. Display the initialized attribute values of that particular object in the main function.

```
//code of the header files are below.
#include<iostream>
#include "emp.h"
using namespace std;

int main()
{
    using namespace emp;
    E.show();

    return 0;
}

emp.h
namespace emp{
    class employee
    {
    protected:
        int emp_id;
        float emp_salary;

    public:
        employee()
        {
            emp_id = 803;
            emp_salary = 30000;
        }

        void show()
        {
            using namespace std;
            cout<<"Employee ID = "<<emp_id<<endl;
            cout<<"Employee Salary = "<<emp_salary<<endl;
        }
    }E;
}
```

35.Write a c++ program where one user defined namespace is embedded within another.

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

namespace outer
{
    int x = 3;
    namespace inner
    {
        int y = 27;
    }
}

int main()
{
    using namespace outer;
    cout<<"Result = "<<x + inner::y<<endl;

    return 0;
}
```

36.How can you declare and define a const object for a given class with protected attributes of type char and int? Provide appropriate method initialize the attributes and display the result.

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

class A
{
    int x;
    char y;

public:
    A(int i, char j)
    {
        x = i;
        y = j;
    }

    void show() const
    {
        cout<<"x = "<<x<<endl;
        cout<<"y = "<<y<<endl;
    }
};

int main()
{
    A const ob(3, 'P');
    ob.show();

    return 0;
}
```

37.Create a text file named as Invoice. The file contains at least 10 item name, code and unit price in rupees. After creating and entering values into that file the contents of same file should be displayed.

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

int main()
{char s[100];
fstream myfile;
 myfile.open("invoice.txt", ios::out);
 string name;
 int code;
 float price;
 for(int i = 0; i < 3; i++)
 {
 cin >> name >> code >> price;
 myfile << name << " " << code << " " << price << " " << endl;
 }
 myfile.close();
myfile.open("invoice.txt", ios::in);
while(!myfile.eof())
{

myfile.getline(s,100);
cout << s;
            cout << endl;

}
  myfile.close();

return 0;
}
```

38.Write a c++ program to create a user defined function that will accept a string as its only parameter. The function is called from the main() and passed a name as a parameter to it to create a binary file with that name.
 In that file at least 30 characters need to be stored. Provide a random access mechanism to access the character at 20th location in that file and display the character at that location. After that the same character need to be overwritten by the letter ‘Z’.

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

int main ()
```

```
{
  fstream fp;
  char ch;
  char s[]="abcdefhijklmnopgrstuvwxyZABCDEFGHIJKLMNOP";
  fp.open("mine.dat", ios::in ios::outlook::binary):
  fp.write(s, sizeof(s));
  cout<<"Original string: "<<s;
  char i[50];
  fp.seekg (19);
  fp.read(&ch, sizeof(char));
  cout<<"\n20th Character read: "<<ch;
  ch='z';
  fp.seekg(19)
  fp.write (& ch, sizeof (ch));
  fp. seekg(0);
  fp.read (i, sizeof (s)); cout<<"\nUpdated string is: "<<i
  <<"\n20th character an updated string: "<<i[19];
  fp.close();
  return 0;
}
```

40.Create a file named as BookDetails. The file should contain information regarding a book like book_id, book_name, year_of_publication. At least five book details should be stored in the above file.
Information regarding the book details should be read and written in bulk mode (use write() and read() methods).

```
#include<iostream>
#include<string>
#include<fstream>
#include<cstring>

using namespace std;

class book{
public:
    char book_id[60];
    char book_name[20];
    int year_of_pub;
};

void disp(book &B)
{cout<<"\n\n\nBook Id "<<B.book_id;
 cout<<"\nBook Name "<<B.book_name;
 cout<<"\nYear of Publication "<<B.year_of_pub;

}
```

```
int main()
{ book A;
  fstream fin;
  fin.open("BookDetails.dat",ios::in|ios::app|ios::binary);

 int n;
 cout<<"\n\nhow many rec. to enter";
 cin>>n;
for(int j=0;j<n;j++)
{

  cout<<"enter the name of the book"<<endl;
cin>>A.book_name;
cout<<"enter the id of the book"<<endl;
cin>>A.book_id;
cout<<"enter the year of publication"<<endl;
cin>>A.year_of_pub;
fin.write((char*)&A,sizeof(book));
}

  fin.seekg(0,ios::beg);
  while(!fin.eof())
  {if(!fin.read((char*)&A,sizeof(A)))break;
  disp(A);
  }
fin.close();


 return 0;
}
```