Assume you have the following declarations:

#include <iostream>

#include <string>

using namespace std;

//////////////////////////////////////////////////

class staff {

private :

 string name ;

 string address;

public:

 staff(string n , string ad) { name=n; address = ad; }

 virtual void display(){cout <<"name = " <<name <<" , address = " <<address;}

 };

//=============================================

class SalaryEmployee : public staff {

private :

 int salary ;

public:

 SalaryEmployee (int s , string n , string ad) :staff(n,ad)

 { salary = s;}

 void display () { staff::display(); cout<<" salary is = " <<salary <<endl;}

};

//=============================================

class HourlyEmployee : public staff{

 private :

 int hourly\_pay ;

public :

 HourlyEmployee (int h , string n , string ad) :staff(n,ad){hourly\_pay=h;}

 void display () { staff::display(); cout<<" hourly\_pay is = " <<hourly\_pay <<endl;}

};

//=============================================

class ContractEmployee

{

};

//=============================================

//print function

///////////////////////////////////

void main (){

}

1. Create a class called **ContractEmployee** that inherits class Staff as public and has **contract duration** as a private element.
	1. The constructor for this class should call the constructor of class Staff as well as initialize its new private elements from the arguments passed to it.
	2. The display function of this class is to call Staff’s display function. And display **contract duration**

**In main**

1. Create an array of 6 pointers to class Staff.
	1. Dynamically create 2 objects of class **SalaryEmployee**, 2 objects of class **HourlyEmployee** and 2 objects of class **ContractEmployee** . ***fill the objects with information from user*** as shown below .

HINT: You will need 3 for loops to fill the array

* 1. Finally write a **function** (**regular function not in a class**) that prints a report of the 6 objects.

The function will use **only 1 for loop** and go over the array one by one each time calling the object’s display function. It does not matter the type of object the array component points to since the pointer can point to all types. This is the advantage of polymorphism.