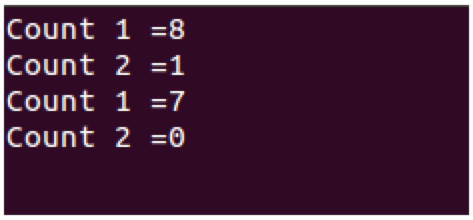
```
/* THIS C++ PROGRAM ILLUSTRSTES THE CONCEPT OF
 * SINGLE INHERITANCE WITH DERIVED CLASS CONSTRUCTOR */

/* NAME : SAGAR GIRI, ROLL : 205, SECTION : A */

#include <iostream>
using namespace std;
class Counter                           //base class
{
      protected:
            unsigned int count;
      public:
            Counter() : count(0)    //base class default constructor
            {           }
            Counter(int c)                //base class one argument constructor
            {     count = c;  }
            Counter operator ++()
            {     return Counter(++count);      }
            int getCount()
            {     return count;     }
};

class CountDn : public Counter            //publicly derived class from base
class
{
      public:
            CountDn():Counter(0)    //derived class default constructor calling
base class default constructor
            {           }
            CountDn(int x):Counter(x)   //derived class one argument constructor
            {           }                             //calling base class one
argument constructor
            Counter operator--()
            {     return Counter(--count);      }
};

int main()
{
      CountDn C1(5),C2;  //define ojects of derived class
      ++C1;++C1;++C1;
      ++C2;
      cout<<"Count 1 ="<<C1.getCount()<<endl;
      cout<<"Count 2 ="<<C2.getCount()<<endl;
      --C1;--C2;
      cout<<"Count 1 ="<<C1.getCount()<<endl;
      cout<<"Count 2 ="<<C2.getCount()<<endl;
return 0;
}
```

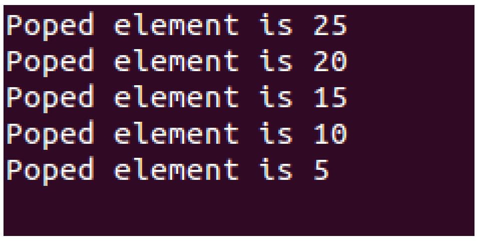
OUTPUT:



```
Count 1 =8
Count 2 =1
Count 1 =7
Count 2 =0
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF
 * FUNCTION OVERRIDING */

/* NAME : SAGAR GIRI, ROLL : 205, SECTION : A */

#include <iostream>
#include <stdlib.h>           //for exit() function
using namespace std;
class Stack                   //base class
{
      protected:
            enum{MAX=5};
            int stack[MAX];
            int top;
      public:
            Stack()
            { top = -1; }
            void push(int var)           //push value into stack
            { stack[++top] = var; }
            int pop()                    //pop value from stack
            { return (stack[top--]); }
};
class FullStack: public Stack //define derived class FullStack from Stack base
                              //class
{
      public:
            void push(int var)           //overriding push function
            {
                  if(top >= MAX-1)
                  {
                        cout<<"Stack Overflow"; exit(1);
                  }
                  Stack::push(var); //call push() method from base class
            }
            int pop()                    //overriding pop function
            {
                  if(top < 0)
                  {
                        cout<<"Stack Underflow";exit(1);
                  }
                  return(Stack::pop());  //call pop() method from base class
            }
};
int main()
{
      FullStack s1;           //define object s1 of derived class
      s1.push(5);             //call push method of derived class
      s1.push(10);
      s1.push(15);
      s1.push(20);
      s1.push(25);
      /*s1.push(30);*/        //shows stack overflow

      cout<<"Poped element is "<<s1.pop()<<endl; //call pop method from derived
class
      cout<<"Poped element is "<<s1.pop()<<endl;
      cout<<"Poped element is "<<s1.pop()<<endl;
      cout<<"Poped element is "<<s1.pop()<<endl;
      cout<<"Poped element is "<<s1.pop()<<endl;
      /*cout<<"Poped element is "<<s1.pop()<<endl;*/ //shows stack underflow
return 0;
}
```

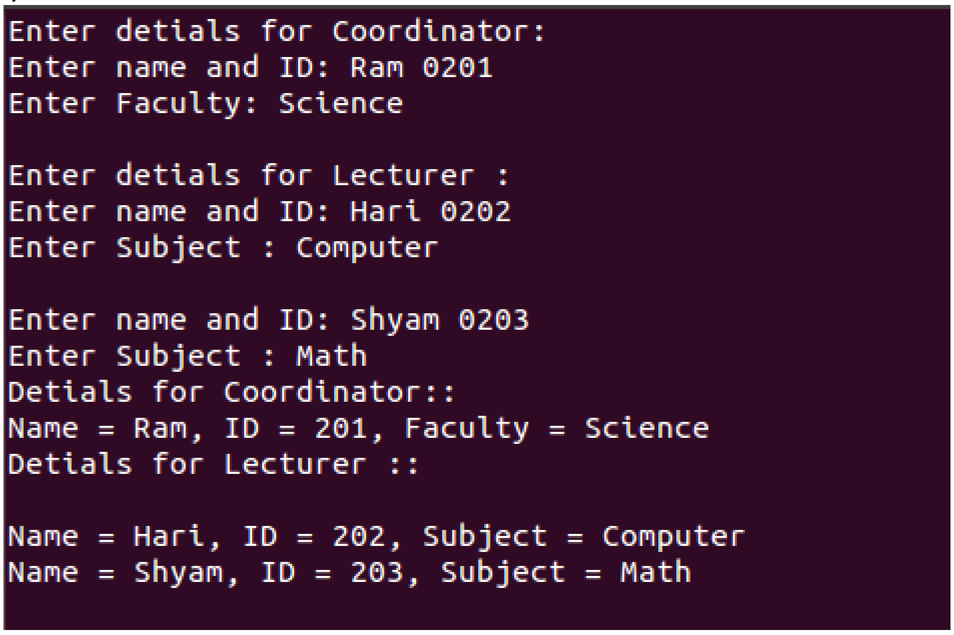
OUTPUT:



```
Poped element is 25
Poped element is 20
Poped element is 15
Poped element is 10
Poped element is 5
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF HIERARCHIAL INHERITANCE */

/* NAME : SAGAR GIRI, ROLL : 205, SECTION : A */

#include <iostream>
using namespace std;
enum{MAX = 10};
class Employee                //define base class employee
{
      private:
            char name[MAX];
            int ID;
      public:
            void getData()
            {
                  cout<<endl<<"Enter name and ID: ";
                  cin>>name>>ID;
            }
            void showData()
            {
                  cout<<endl<<"Name = "<<name<<", ID = "<<ID;
            }
};
//derived class Coordinator from employee class
class Cordinator: private Employee
{
      private:
            char faculty[MAX];
      public:
            void getData()
            {
                  Employee::getData();
                  cout<<"Enter Faculty: ";
                  cin>>faculty;
            }
            void showData()
            {
                  Employee::showData();
                  cout<<", Faculty = "<<faculty;
            }
};
//derive Lecturer class from Employee class
class Lecturer : private Employee
{
      private:
            char subject[MAX];
      public:
            void getData()
            {
                  Employee::getData();
                  cout<<"Enter Subject : ";
                  cin>>subject;
            }
            void showData()
            {
                  Employee::showData();
                  cout<<", Subject = "<<subject;
            }
};

int main()
{
        Cordinator c1;          //object c1 of Coordinator class
        Lecturer l1,l2;         //Object l1,l2 of Lecturer class
        cout<<"Enter detials for Coordinator: ";c1.getData();
        cout<<endl<<"Enter detials for Lecturer : ";
        l1.getData();
        l2.getData();

        cout<<"Detials for Coordinator:: ";
        c1.showData();
        cout<<"\nDetials for Lecturer :: \n";
        l1.showData();
        l2.showData();
return 0;
}
```

//OUTPUT

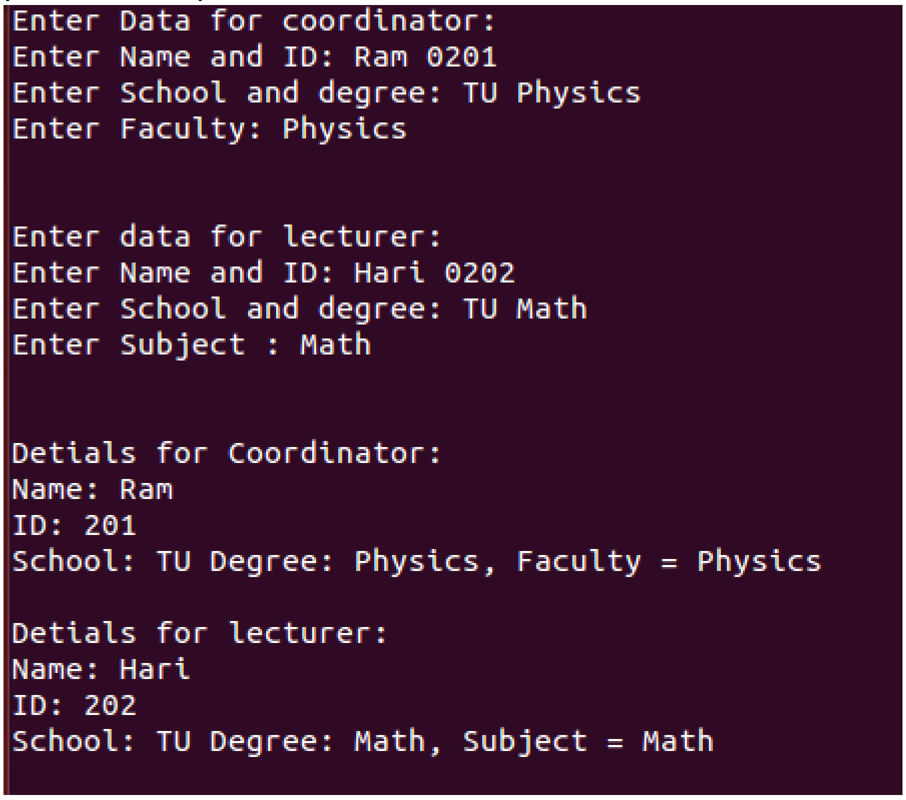


```
Enter detials for Coordinator:
Enter name and ID: Ram 0201
Enter Faculty: Science

Enter detials for Lecturer :
Enter name and ID: Hari 0202
Enter Subject : Computer

Enter name and ID: Shyam 0203
Enter Subject : Math
Detials for Coordinator::
Name = Ram, ID = 201, Faculty = Science
Detials for Lecturer ::

Name = Hari, ID = 202, Subject = Computer
Name = Shyam, ID = 203, Subject = Math
```

```
/* THIS C++ PROGRAM ILLUSTRATED THE CONCEPT OF MULTIPLE INHERITANCE */

/* NAME : SAGAR GIRI, ROLL : 205, SECTION : A */

#include <iostream>
using namespace std;
class Employee               //define base class Employee
{
      private:
            char name[20];
            int ID;
      public:
            void getData()
            {
                  cout<<endl<<"Enter Name and ID: ";
                  cin>>name>>ID;
            }
            void showData()
            {
                  cout<<endl<<"Name: "<<name<<endl<<"ID: "<<ID;
            }
};
class Education              //define another base class Education
{
      private:
            char school[20];
            char degree[20];
      public:
            void getData()
            {
                  cout<<"Enter School and degree: ";
                  cin>>school>>degree;
            }
            void showData()
            {
                  cout<<endl<<"School: "<<school<<" Degree: "<<degree;
            }
};

//Define derived class Coordinator derived from base class Employee & Education
class Coordinator:private Employee, private Education
{
      private:
                  char faculty[20];
            public:
                  void getData()
                  {
                        Employee::getData();
                        Education::getData();
                        cout<<"Enter Faculty: ";
                        cin>>faculty;
                  }
                  void showData()
                  {
                        Employee::showData();
                        Education::showData();
                        cout<<", Faculty = "<<faculty;
                  }
};
//Define derived class Lecturer derived from base class Employee & Education
class Lecturer : private Employee, private Education
{
```

```
            private:
                  char subject[20];
            public:
                  void getData()
                  {
                        Employee::getData();
                        Education::getData();
                        cout<<"Enter Subject : ";
                        cin>>subject;
                  }
                  void showData()
                  {
                        Employee::showData();
                        Education::showData();
                        cout<<", Subject = "<<subject;
                  }
};
int main()
{
      Coordinator c1;         //define object c1 of Coordinator class
      Lecturer l1;            //define object l1 of Lecturer class

      cout<<"Enter Data for coordinator: ";
      c1.getData();
      cout<<endl<<endl<<"Enter data for lecturer: ";
      l1.getData();

      cout<<endl<<endl<<"Detials for Coordinator: ";
      c1.showData();

      cout<<endl<<endl<<"Detials for lecturer: ";
      l1.showData();
return 0;
}
```

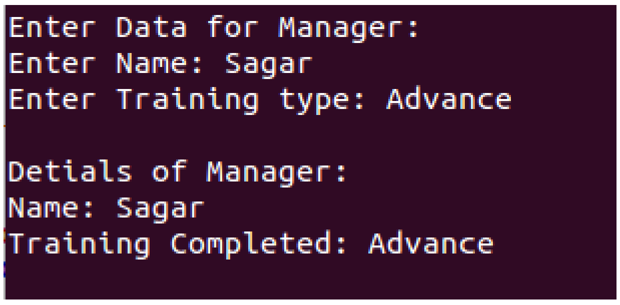
/* OUTPUT: */



```
Enter Data for coordinator:
Enter Name and ID: Ram 0201
Enter School and degree: TU Physics
Enter Faculty: Physics


Enter data for lecturer:
Enter Name and ID: Hari 0202
Enter School and degree: TU Math
Enter Subject : Math


Detials for Coordinator:
Name: Ram
ID: 201
School: TU Degree: Physics, Faculty = Physics

Detials for lecturer:
Name: Hari
ID: 202
School: TU Degree: Math, Subject = Math
```

```cpp
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF AMBUIGITY
 * ASSOCIATED WITH THE MULTIPLE INHERITANCE */

/* NAME : SAGAR GIRI, ROLL : 205, SECTION : A */

#include <iostream>
using namespace std;
class Employee                      //define base class Employee
{
      protected:
            char name[20];
      public:
            void getName()
            {
                  cout<<endl<<"Enter Name: ";
                  cin>>name;
            }
            void showData()
            {
                  cout<<endl<<"Name: "<<name;
            }
};
class Training                      //define base class Training
{
      protected:
            char type[20];
      public:
            void getData()
            {
                  cout<<"Enter Training type: ";
                  cin>>type;
            }
            void showData()
            {
                  cout<<endl<<"Training Completed: "<<type;
            }
};
//derived class Manager from base class Employee and Training
class Manager: public Employee, public Training
{
      public:
            void getData()
            {
                  Employee::getName();
                  Training::getData();
            }
};

int main()
{
      Manager m1;                   //define object m1 of Manager class
      cout<<"Enter Data for Manager: ";
      m1.getData();

      cout<<endl<<"Detials of Manager: ";
      /* m1.showData(); */    //compiler generates error due to ambiguity

      m1.Employee::showData();      //call showData() methof from Employee
      m1.Training::showData();      //call showData() methof from Training
return 0;
}
```

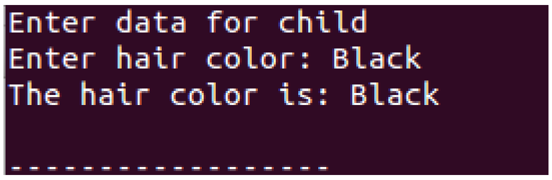
OUTPUT:



```
Enter Data for Manager:
Enter Name: Sagar
Enter Training type: Advance

Detials of Manager:
Name: Sagar
Training Completed: Advance
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF AMBUIGUITY ASSOCIATED
 * WITH THE MULTIPATH INHERITANCE */

/* NAME : SAGAR GIRI, ROLL : 205, SECTION : A*/

#include <iostream>
using namespace std;
class Grandfather                    //define base class Grandfather
{
      protected:
            char hairColor[10];
      public:
            void getData()
            {
                  cout<<"Enter hair color: ";
                  cin>>hairColor;
            }
            void showData()
            {
                  cout<<"The hair color is: "<<hairColor;
            }
};
/* class Father: public Grandfather{}; */ //Compiler generates error due to
ambuiguity

class Father: virtual public Grandfather
{                 };

/* class Mother: public Grandfather{}; */ //Compiler generates error due to
ambuiguity

class Mother: virtual public Grandfather
{                 };

class Child: public Father, public Mother //derived class Child from class
Father & Mother
{                 };

int main()
{
      Child c1;                     //Object c1 of class Child
      cout<<"Enter data for child"<<endl;
      c1.getData();           //calls getData() from Grandfather class
      c1.showData();          //calls showData() from Grandfather class
return 0;
}
```

OUTPUT:



```
Enter data for child
Enter hair color: Black
The hair color is: Black

------------------
```