```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF
 * PURE VIRTUAL CLASS, ABSTRACT CLASS AND LATE
 * BINDING IN THE POLYMORPHISM*/

 /* NAME: SAGAR GIRI, ROLL: 205 , SECTION : A */
#include <iostream>
using namespace std;
class Base        //ABSTRACT CLASS
{
      public:
            virtual void show() = 0;      //PURE VIRTUAL FUNCTION
};
class Derived1 : public Base
{
      public:
            void show()
            {
                  cout<<"This is Derived1..."<<endl;
            }
};
class Derived2 : public Base
{
      public:
            void show()
            {
                  cout<<"This is Derived2..."<<endl;
            }
};
int main()
{
      Base *ptr;
      Derived1 dv1;
      Derived2 dv2;
      cout<<"Enter the choice 1 or 2"<<endl;
      int c;
      cin>>c;

      //THE OUTPUT DEPENDS ON THE USER INPUT i.e. IN RUN-TIME
      if(c == 1)
      {
            ptr = &dv1;
      }
      else
      {
            ptr = &dv2;
      }
      ptr -> show();
return 0;
}
```

OUTPUT:

```
Enter the choice 1 or 2
1
This is Derived1...
```

```
Enter the choice 1 or 2
2
This is Derived2...
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF VIRTUAL
 * FUNCTION AND EARLY BINDING OF STATIC POLYMORPHISM*/

 /* NAME: SAGAR GIRI, ROLL: 205, SECTION: A*/
#include <iostream>
using namespace std;
class Base  //ABSTRACT CLASS
{
      public:
            virtual void show()
            {
                  cout<<"This is base..."<<endl;
            }
};
class Derived1 : public Base
{
      public:
            void show()
            {
                  cout<<"This is derived one..."<<endl;
            }
};
class Derived2 : public Base
{
      public:
            void show()
            {
                  cout<<"This is derived two..."<<endl;
            }
};
int main()
{
      Base *ptr; //BASE CLASS POINTER

      /* BASE CLASS POINTER COULD HOLD THE
       * ADDRESS OF DERIVED CLAAS'S OBJECT
       * AND VICE-VERSA IS NOT POSSIBLE*/

      Derived1 dv1;
      Derived2 dv2;
      ptr = &dv1;
      ptr -> show();

      ptr = & dv2;
      ptr -> show();
return 0;
}
```

OUTPUT:

```
This is derived one...
This is derived two...
```

```
/* THIS C++ PROGRAM ILLUSTRATES THE CONCEPT OF STATIC FUNCTIONS
 * AND A DESTRUCTOR */

/* NAME: SAGAR GIRI, ROLL: 205, SECTION: A*/
#include <iostream>
using namespace std;
class gamma
{
      private:
            static int total;
            int id;
      public:
            gamma()                 //DEFAULT CONSTRUCTOR
            {
                  total++;
                  id = total;
            }
            ~gamma()          //DESTRUCTOR
            {
                  total--;
                  cout<<"Destroying ID number "<<id<<endl;
            }
            static void showTotal() //STATIC FUNCTION
            {
                  cout<<"Total is : "<<total<<endl;
            }
            void showID()           //NON-STATIC FUNCTION
            {
                  cout<<"ID number is : "<<id<<endl;
            }
};
int gamma::total = 0;

int main()
{
      gamma g1;
      //STATIC FUNCTION ARE ASSOCIATED WITH THE CLASS ITSELF
      gamma::showTotal();

      gamma g2,g3;
      gamma::showTotal();

      //NON-STATIC FUNCTIONS ARE ASSOCIATED WITH THE OBJECTS ONLY
      g1.showID();
      g2.showID();
      g3.showID();

      cout<<"<-----End Of Program------>"<<endl;
//DESTRUCTORS ARE CALLED JUST BEFORE THE RETURN STATEMENT
return 0;
}
```

OUTPUT:

```
Total is : 1
Total is : 3
ID number is : 1
ID number is : 2
ID number is : 3
<-----End Of Program------>
Destroying ID number 3
Destroying ID number 2
Destroying ID number 1
```