* ***Purpose :*** **Classwork.**

* ***Date :*** **02/01/2026**

* ***Author :*** **Vikas Srivastava**

* ***ID :*** **55984**

* ***Batch ID :*** **25SUB4505**

---

1. ***Code :***

virtFunctionsTwo.cpp

```cpp
// Purpose - Write a program to demonstrate runtime polymorphism in C++ using virtual functions, showing how a base class pointer can call overridden functions in derived class and ensuring proper destruction with a virtual destructor.

#include <iostream>
using namespace std;

class Base{
public:
  Base(){cout<<"Base::Base() --> ctor"<<endl;}
  virtual ~Base(){cout<<"Base::~Base() --> dtor"<<endl;}
  virtual void funOne(){cout<<"Base::funOne()"<<endl;}
  virtual void funTwo(){cout<<"Base::funTwo()"<<endl;}
};

class Derived:public Base{
public:
  Derived(){cout<<"Derived::Derived() --> ctor"<<endl;}
  ~Derived(){cout<<"Derived::~Derived()--> dtor"<<endl;}

  void funOne(){cout<<"Derived::funOne()"<<endl;}
  void funTwo(){cout<<"Derived::funTwo()"<<endl;}
};

int main(){
  Base *bPtr = new Base();
  bPtr->funOne();
  bPtr->funTwo();
  delete bPtr;
  cout<<"*********************************************\n";

  bPtr = new Derived();
  bPtr->funOne();
  bPtr->funTwo();
  delete bPtr;
}
```

Output :

```
Base::Base() --> ctor
Base::funOne()
Base::funTwo()
Base::~Base()--> dtor
*********************************************
Base::Base() --> ctor
Derived::Derived() --> ctor
Derived::funOne()
Derived::funTwo()
Derived::~Derived()--> dtor
Base::~Base()--> dtor
```

2. ***Code :***

virtFunctionsOne.cpp

```cpp
// Purpose - Write a program to demonstrate runtime polymorphism in C++ using virtual functions, showing dynamic dispatch and proper destruction of objects through a base class pointer.

#include <iostream>
using namespace std;

class Base{
public:
  Base(){cout<<"Base::Bas
  virtual ~Base(){cout<<"Base::~Base()--> dtor"<<endl;}
  virtual void funOne(){cout<<"Base::funOne()"<<endl;}
  virtual void funTwo(){cout<<"Base::funTwo()"<<endl;}
};

class Derived:public Base{
public:
  Derived(){cout<<"Derived::Derived() --> ctor"<<endl;}
  ~Derived(){cout<<"Derived::~Derived()--> dtor"<<endl;}

  void funOne(){cout<<"Derived::funOne()"<<endl;}
  void funTwo(){cout<<"Derived::funTwo()"<<endl;}
};

void funCaller(Base *bptr){
  bptr->funOne();
  bptr->funTwo();
  delete bptr;
  cout<<"*********************************************\n";
}

int main(){
  funCaller(new Base());
  funCaller(new Derived());
}
```

(const char [22])"Base::~Base()--> dtor"
Loading...

Output :

```
Base::Base() --> ctor
Base::funOne()
Base::funTwo()
Base::~Base()--> dtor
*********************************************
Base::Base() --> ctor
Derived::Derived() --> ctor
Derived::funOne()
Derived::funTwo()
Derived::~Derived()--> dtor
Base::~Base()--> dtor
*********************************************
```

3. Code :

problemStdAttend.cpp

```cpp

/*
 * Purpose: An attendance system manages students of different programs: Engineering,
 * Arts, and Science. Each student has a name and attendance percentage.
 */
#include <iostream>
using namespace std;

class StdAttendance{
protected:
  string name;
  double attendance;
public:
  StdAttendance(const string &n, double attn):name(n), attendance(attn){}
  virtual bool checkEligibility() = 0;
  string getName(){return name;}
};

class EngineeringStudent: public StdAttendance{
public:
  EngineeringStudent(const string &n, double att): StdAttendance(n, att) {}
  bool checkEligibility(){
    return attendance >= 75;
  }
};
class ArtsStudent: public StdAttendance{
public:
  ArtsStudent(const string &n, double att): StdAttendance(n, att) {}
  bool checkEligibility(){
    return attendance >= 65;
  }
};
class ScienceStudent: public StdAttendance{
public:
  ScienceStudent(const string &n, double att): StdAttendance(n, att) {}
  bool checkEligibility(){
    return attendance >= 70;
  }
```

problemStdAttend.cpp

```cpp
class ScienceStudent: public StdAttendance{
public:
  ScienceStudent(const string &n, double att): StdAttendance(n, att) {}
  bool checkEligibility(){
    return attendance >= 70;
  }
};


int main(){
  StdAttendance *sArr[] = {new ArtsStudent("Smitha", 68), new ScienceStudent("Rahul", 72), new EngineeringStudent("Sachin", 78), new ArtsStudent("Balu",
  64), new ScienceStudent("Sharma", 72),new EngineeringStudent("Rathore", 79), nullptr};


  for(int cnt =0 ;sArr[cnt] != nullptr; cnt++){
    cout<<sArr[cnt]-> getName()<<" is ";
    if( sArr[cnt]->checkEligibility())
      cout<<"Eligible"<<endl;
    else
      cout<<"Not Eligible"<<endl;
  }


  /*

  StdAttendance *s = new EngineeringStudent("Rahul", 78);
  cout<<s->getName()<<" is ";
  if( s->checkEligibility())
    cout<<"Eligible"<<endl;
  else
    cout<<"Not Eligible"<<endl;
  */
}
```

Output :

```
Smitha is Eligible
Rahul is Eligible
Sachin is Eligible
Balu is Not Eligible
Sharma is Eligible
Rathore is Eligible
```