

\* Purpose : Classwork.

\* Date : 29/12/2025

\* **Author : Vikas Srivastava**

\* ID : 55984

\* Batch ID : 25SUB4505

1. **Code :** Write a program to implement a custom MyArray class in C++ that demonstrates abstraction, constructors (default, parameterized, copy constructor), and member functions to initialize and display array elements, showcasing object-oriented programming principles.

```

1  #include <iostream>
2  using namespace std;
3  class MyArray { //abstraction
4      int arr[100];
5      int size;
6  public:
7      MyArray();
8      MyArray(int data);
9      MyArray(const MyArray&);
10     void printArray();
11     MyArray& extend(MyArray&);
12 };
13 int main(){
14     MyArray objOne;
15     MyArray objTwo(100);
16     MyArray objThree = objTwo;
17     objOne.printArray();
18     objTwo.printArray();
19     objThree.printArray();
20     //objFour = objTwo.extend(objThree);
21     //objFour.printArray();
22 }
23 MyArray::MyArray(){
24     size = 100;
25     for (int cnt=0;cnt< size; cnt++)
26         arr[cnt] = 0;
27 }
28 MyArray::MyArray(int data){
29     size = 100;
30     for (int cnt=0;cnt< size; cnt++)
31         arr[cnt] = data + cnt;
32 }

```

```

13  int main(){
14      //objFour = objTwo.extend(objThree);
15      //objFive = objThree.extend(objFour);
16  }
17  MyArray::MyArray(){
18      size = 100;
19      for (int cnt=0;cnt< size; cnt++)
20          arr[cnt] = 0;
21  }
22  MyArray::MyArray(int data){
23      size = 100;
24      for (int cnt=0;cnt< size; cnt++)
25          arr[cnt] = data + cnt;
26  }
27  MyArray::MyArray(const MyArray& rhsObj){
28      size = rhsObj.size;
29      for (int cnt=0;cnt< size; cnt++)
30          arr[cnt] = rhsObj.arr[cnt];
31  }
32  void MyArray::printArray(){
33      cout<<"Size: "<<size<<"\n";
34      for(int cnt=0;cnt < size; cnt++)
35          cout<<arr[cnt]<<" ";
36      cout<<endl;
37  }
38  MyArray& MyArray::extend(MyArray& rhsObject){
39      //For future implementation with dynamic memory allocation
40      return rhsObject;
41  }

```

**Output :**

[illegible]

2. **Code :** Write a program to demonstrate the invocation order of constructors and destructors in C++, including the default constructor, parameterized constructor, and copy constructor, and observe how the destructor is called automatically when objects go out of scope.

```
copyCtorOne.cpp X
copyCtorOne.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 class Test{
5 public:
6     Test(){cout<<"Test()--> Default Ctor"<<endl; }
7     Test(int){cout<<"Test(int)--> Parameterized Ctor"<<endl; }
8
9     ~Test(){cout<<"~Test()--> Destructor"<<endl; }
10    Test(const Test &) {cout<<"Test(const Test&) ---> Copy Ctor"<<endl;}
11 };
12
13
14 int main(){
15     Test objOne ;//single parameterized ctor
16     Test objTwo = 100;//single parameterized ctor
17     Test objThree = objTwo;//single parameterized ctor
18 }
```

**Output :**

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\copyCtorOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Test()--> Default Ctor
Test(int)--> Parameterized Ctor
Test(const Test&) ---> Copy Ctor
~Test()--> Destructor
~Test()--> Destructor
~Test()--> Destructor
```

3. **Code :** Write a program to demonstrate passing an object by reference to a function in C++, showing that no copy constructor is called when objects are passed by reference, and only the existing object is accessed and returned.

```
copyCtorThree.cpp X
copyCtorThree.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 class Test{
5 public:
6     Test(){cout<<"Test()--> Default Ctor"<<endl; }
7     Test(int){cout<<"Test(int)--> Parameterized Ctor"<<endl; }
8
9     ~Test(){cout<<"~Test()--> Destructor"<<endl; }
10    Test(const Test &) {cout<<"Test(const Test&) ---> Copy Ctor"<<endl;}
11 };
12
13 Test& fun(Test& arg){ //call by reference
14     return arg;
15 }
16
17
18 int main(){
19     Test objTwo = 100;//single parameterized ctor
20
21     fun(objTwo);
22 }
```

**Output :**

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\copyCtorThree.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Test(int)--> Parameterized Ctor
~Test()--> Destructor
```

4. **Code :** Write a program to demonstrate passing an object by value to a function in C++, where a copy of the object is created, resulting in the copy constructor being invoked, and the copied object is destroyed when it goes out of scope.

```
copyCtorTwo.cpp X
copyCtorTwo.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 class Test{
5 public:
6     Test(){cout<<"Test()--> Default Ctor"<<endl; }
7     Test(int){cout<<"Test(int)--> Parameterized Ctor"<<endl; }
8
9     ~Test(){cout<<"~Test()--> Destructor"<<endl; }
10    Test(const Test &) {cout<<"Test(const Test&) ---> Copy Ctor"<<endl;}
11 };
12
13 Test fun(Test arg){//call by value
14
15     return arg;
16 }
17
18 int main(){
19     Test objTwo = 100;//single parameterized ctor
20
21     fun(objTwo);
22 }
```

**Output :**

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\copyCtorTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Test(int)--> Parameterized Ctor
Test(const Test&) ---> Copy Ctor
Test(const Test&) ---> Copy Ctor
~Test()--> Destructor
~Test()--> Destructor
~Test()--> Destructor
```

5. **Code :** Write a program to demonstrate deep copy in C++ using a copy constructor, where dynamically allocated memory is duplicated for each object so that both objects manage separate memory, preventing issues like double deletion and dangling pointers.

```
DeepCopy.cpp X
DeepCopy.cpp > Resource
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 class Resource {
6 public:
7     char* ptr;
8
9     // Constructor
10    Resource(const char* str) {
11        ptr = new char[strlen(str) + 1];
12        strcpy(ptr, str);
13        cout << "Memory allocated: " << ptr << endl;
14    }
15
16    // Deep Copy Constructor
17    Resource(const Resource& other) {
18        ptr = new char[strlen(other.ptr) + 1];
19        strcpy(ptr, other.ptr);
20        cout << "Deep copy created: " << ptr << endl;
21    }
22
23    // Destructor
24    ~Resource() {
25        cout << "Deleting memory: " << ptr << endl;
26        delete[] ptr;
27    }
28 };
29
30 int main() {
31     Resource r1("Network");
32     Resource r2 = r1; // Deep copy now
33
34     return 0;
35 }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\DeepCopy.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Memory allocated: Network
Deep copy created: Network
Deleting memory: Network
Deleting memory: Network
```

6. **Code :** Write a program to demonstrate the concept of a friend class in C++, where one class (UsingTest) is declared as a friend of another class (Test), allowing it to access the private data members and member functions of the class, illustrating controlled access and encapsulation.

```
friendClassOne.cpp X
friendClassOne.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  //class UsingTest;
5
6  class Test{//all members of a class is private by default
7      int data ;
8      Test(int x=10): data(x){ cout <<"constructor called"<<endl;}
9      void disp(){cout<<"Data: "<<data<<endl; }
10     friend class UsingTest;
11 };
12
13 class UsingTest{
14     Test obj;
15 public:
16     UsingTest(int x=100): obj(Test(x)) {}
17     void print(){ obj.disp() ; }
18 };
19
20 int main(){
21     UsingTest obj=100;
22     obj.print();
23 }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\friendClassOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
constructor called
Data: 100
```

7. **Code :** Write a program to demonstrate the use of a friend function in C++, where the main() function is declared as a friend of a class, allowing it to access the class's private constructor and private member function, illustrating controlled access beyond normal encapsulation rules.

```
friendOne.cpp X
friendOne.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  class Test{//all members of a class is private by default
5      int data ;
6      Test(int x=10): data(x){ cout <<"constructor called"<<endl;}
7      void disp(){cout<<"Data: "<<data<<endl; }
8
9      friend int main();
10 };
11
12 int main(){
13     Test obj=100;
14     obj.disp();
15 }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\friendOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
constructor called
Data: 100
```

8. **Code :** Write a program to demonstrate the use of a friend function in C++, where a non-member function is granted access to the private data members of a class to display internal object information.

```
friendTwo.cpp X
friendTwo.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  class Box{
5  |   int length;
6  |   public:
7  |       Box(int len = 10): length(len) {}
8  |
9  |       friend void showLength(Box &b);
10 |   };
11
12 |   void showLength(Box &b){
13 |       cout<<"Length : "<<b.length<<endl;
14 |   }
15
16 |   int main(){
17 |       Box gift;
18 |       showLength(gift);
19 |   }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\friendTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Length : 10
```

9. **Code :** Write a program to demonstrate single inheritance in C++, where a derived class inherits public member functions from a base class and accesses them through a derived class object.

```
inherOne.cpp X
inherOne.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  class Base{
5  |   public:
6  |       void funOne(){cout<<"Base::funOne()"<<endl; }
7  |       void funTwo(){cout<<"Base::funTwo()"<<endl; }
8  |   };
9
10 |   class Derived:public Base{};
11
12 |   int main(){
13 |       Derived dObj;
14 |       dObj.funOne();
15 |       dObj.funTwo();
16 |   }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\inherOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Base::funOne()
Base::funTwo()
```

10. **Code** : Write a program to implement a custom array class (MyArray) in C++ that demonstrates abstraction, default constructor, parameterized constructor, and copy constructor, along with a member function to display array elements, illustrating object creation and copying behavior.

```

1  #include <iostream>
2  using namespace std;
3
4  class MyArray{ //abstraction
5      int arr[100];
6      int size;
7  public:
8      MyArray();
9      MyArray(int data);
10     MyArray(const MyArray&);
11     void printArray();
12     MyArray& extend(MyArray&);
13 };
14
15
16 int main(){
17     MyArray objOne;
18     MyArray objTwo(100);
19     MyArray objThree = objTwo;
20     objOne.printArray();
21     objTwo.printArray();
22     objThree.printArray();
23
24     //objFour = objTwo.extend(objThree);
25     //objFour.printArray();
26 }
27
28 MyArray::MyArray(){
29     size = 100;
30     for (int cnt=0;cnt< size; cnt++)
31         arr[cnt] = 0;
32 }

```

```

33
34   MyArray::MyArray(int data){
35       size = 100;
36       for (int cnt=0;cnt< size; cnt++){
37           arr[cnt] = data + cnt;
38       }
39
40
41   MyArray::MyArray(const MyArray& rhsObj){
42       size = rhsObj.size;
43       for (int cnt=0;cnt< size; cnt++){
44           arr[cnt] = rhsObj.arr[cnt];
45       }
46   }
47   void MyArray::printArray(){
48       cout<<"Size: "<<size<<"\tarray: "<<endl;
49       for(int cnt=0;cnt < size; cnt++){
50           cout<<arr[cnt]<<" ";
51       cout<<endl;
52   }
53
54   MyArray& MyArray::extend(MyArray& rhsObject){
55       //For future implementation with dynamic memory allocation
56
57       return rhsObject;
58   }

```

**Output :**

[illegible]

11. **Code :** Write a program to demonstrate deep copy in C++ for a class that uses dynamic memory allocation, where a copy constructor allocates separate memory for each object to prevent dangling pointers and double deletion when objects go out of scope.

```
myArrayPtrOne.cpp X
myArrayPtrOne.cpp > ...
1 #include <iostream>
2 using namespace std;
3 //code has an issue - dangling pointer
4 class MyArray{
5     int *arr;
6     int size;
7 public:
8     MyArray();
9     MyArray(int data);
10    MyArray(const MyArray& );
11    void printArray();
12
13    ~MyArray(){
14        if (size){
15            delete [] arr;
16        }
17        size=0;
18    }
19 };
20
21 int main(){
22     MyArray objTwo(100); //single parameterized ctor
23     objTwo.printArray();
24     {
25         MyArray objThree = objTwo; //copying one object to another
26         objThree.printArray();
27     }
28     objTwo.printArray();
29 }
30
31 MyArray::MyArray(const MyArray& rhs){
32     size = rhs.size;
33     arr = new int[size]; //separate memory allocation
34     for (int cnt = 0; cnt < size; cnt++)
35         arr[cnt] = rhs.arr[cnt]; //data being copied
36
37     arr[cnt] = rhs.arr[cnt]; //data being copied
38 }
39
40 MyArray::MyArray():size(10){
41     arr = new int[size];
42     for (int cnt=0; cnt< size; cnt++)
43         arr[cnt] = 0;
44 }
45
46 MyArray::MyArray(int sz):size(sz) {
47     int data = 101;
48     arr = new int[size];
49     for (int cnt=0; cnt< size; cnt++)
50         arr[cnt] = data + cnt;
51 }
52
53 void MyArray::printArray(){
54     cout<<"Size: "<<size<<"\tarray: "<<endl;
55     for(int cnt=0; cnt < size; cnt++)
56         cout<<arr[cnt]<<" ";
57     cout<<endl;
58 }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C++\Day_6\classwork> g++ .\myArrayPtrOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C++\Day_6\classwork> .\a.exe
Size: 100 array:
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152
153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178
179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
Size: 100 array:
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152
153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178
179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
Size: 100 array:
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152
153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178
179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
```

12. **Code :** Write a program to demonstrate the use of the default copy constructor in C++, where an object containing a statically allocated array is copied to another object, resulting in a member-wise (shallow) copy that works safely because no dynamic memory allocation is involved.

```
myArrayThree.cpp X
myArrayThree.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  class MyArray{ //abstraction
5      int arr[100]; //array
6      int size; //integer
7  public:
8      MyArray();
9      MyArray(int data);
10     void printArray();
11 };
12
13 int main(){
14     MyArray objTwo(100); //single parameterized ctor
15     MyArray objThree = objTwo; //copying one object to another
16
17     objTwo.printArray();
18     objThree.printArray();
19 }
20
21 MyArray::MyArray(){
22     size = 100;
23     for (int cnt=0; cnt< size; cnt++)
24         arr[cnt] = 0;
25 }
26
27 MyArray::MyArray(int data){
28     size = 100;
29     for (int cnt=0; cnt< size; cnt++)
30         arr[cnt] = data + cnt;
31 }
32
33 void MyArray::printArray(){
34     cout<<"Size: "<<size<<"\tarray: "<<endl;
35     for(int cnt=0; cnt < size; cnt++)
36         cout<<arr[cnt]<<" ";
37     cout<<endl;
38 }
```

### Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ -I..myArrayThree.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Size: 100      array:
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
Size: 100      array:
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
```



- 13. Code :** Write a program to implement a custom MyArray class in C++ that demonstrates static array usage, default, parameterized, and copy constructors, and a member function to extend (merge) two array objects into a new object, illustrating object copying, array manipulation, and return-by-value behavior.

```
myArrayTwo.cpp X
myArrayTwo.cpp > main()
1 #include <iostream>
2 using namespace std;
3
4 class MyArray{ //abstraction
5     static const int max = 100;
6     int arr[max];
7     int size;
8 public:
9     MyArray(); //size = 10
10    MyArray(int data); //size = 20
11    MyArray(const MyArray&); //copy ctor
12    void printArray();
13    MyArray extend(MyArray&); //10 + 20 --> less than 100(max)
14 };
15
16 int main(){
17     MyArray objOne;
18     MyArray objTwo(100);
19     MyArray objThree = objTwo; //copy constructor called
20     objOne.printArray();
21     objTwo.printArray();
22     objThree.printArray();
23
24     MyArray objFour = objOne.extend(objThree);
25     objFour.printArray();
26 }
27
28 MyArray::MyArray(){
29     size = 10;
30     for (int cnt=0; cnt< size; cnt++)
31         arr[cnt] = 0;
32 }
```

```
myArrayTwo.cpp X
myArrayTwo.cpp > main()
33
34 MyArray::MyArray(int data){
35     size = 20;
36     for (int cnt=0; cnt< size; cnt++)
37         arr[cnt] = data + cnt;
38 }
39
40 MyArray::MyArray(const MyArray& rhsObj){
41     size = rhsObj.size;
42     for (int cnt=0; cnt< size; cnt++)
43         arr[cnt] = rhsObj.arr[cnt];
44 }
45
46 void MyArray::printArray(){
47     cout<<"Size: "<<size<<"\tarray: "<<endl;
48     for(int cnt=0; cnt < size; cnt++)
49         cout<<arr[cnt]<<" ";
50     cout<<endl;
51 }
52
53 MyArray MyArray::extend(MyArray& rhsObj){
54     MyArray newObj;
55     newObj.size = size + rhsObj.size;
56
57     for (int cnt=0; cnt< size; cnt++)
58         newObj.arr[cnt] = arr[cnt];
59
60     for (int nCnt=0, cnt = size; cnt < newObj.size; cnt++)
61         newObj.arr[cnt] = rhsObj.arr[nCnt++];
62
63     return newObj;
64 }
```

### Output :

```
PS C:\Users\VIDAS\Source\STAW\OneDrive\Desktop\IC_OPP\Day_9\Classwork> g++ .\myArrayTwo.cpp
PS C:\Users\VIDAS\Source\STAW\OneDrive\Desktop\IC_OPP\Day_9\Classwork> .\a.exe
Size: 10      array:
0 0 0 0 0 0 0 0 0 0
Size: 20      array:
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
Size: 20      array:
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
Size: 30      array:
0 0 0 0 0 0 0 0 0 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
```

14. **Code :** Write a program to implement the Singleton Design Pattern in C++, where a class ensures that only one object is created throughout the program by using a private constructor, a static pointer, and a static member function to provide controlled access to the single instance.

```
singleTon.cpp
singleTon.cpp > Singleton > ptr2Obj
1  #include <iostream>
2  using namespace std;
3
4  class Singleton{
5  private: //data members here
6      static Singleton *ptr2Obj;
7      Singleton(){
8          cout<<"Object created..."<<endl;
9      }
10     Singleton(int x){
11         cout<<"Object created...with "<<x<<endl;
12     }
13     Singleton(const Singleton &);
14 public:
15     static Singleton& getSingleton(){
16         if (ptr2Obj==nullptr)
17             ptr2Obj = new Singleton();
18         return *ptr2Obj;
19     }
20     void disp(){
21         cout<<"Singleton::disp()"<<endl;
22     }
23 };
24
25
26 Singleton* Singleton::ptr2Obj=nullptr;
27 int main(){
28     //Singleton objOne;//this is an error
29
30     Singleton &refObj = Singleton::getSingleton();
31     refObj.disp();
32 }
```

**Output :**

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\singleTon.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Object created...
Singleton::disp()
```

15. **Code :** Write a program to demonstrate static data members and static member functions in C++, where a class-level variable is shared among all objects and can be accessed using both the class name and object instances.

```
staticOne.cpp X
staticOne.cpp > ...
1  #include <iostream>
2  using namespace std;
3
4  class Employee{
5      static string compName;//class member declaration
6  public:
7      static string company(){return compName; }//class member
8  };
9
10 int main(){
11     cout<<"Employee::company() --> "<< Employee::company()<<endl;
12     Employee obj1;
13     cout<<"obj1.company() --> "<< obj1.company()<<endl;
14 }
15
16 string Employee::compName = "Wipro Ltd";
```

### **Output :**

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\staticOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Employee::company() --> Wipro Ltd
obj1.company() --> Wipro Ltd
```