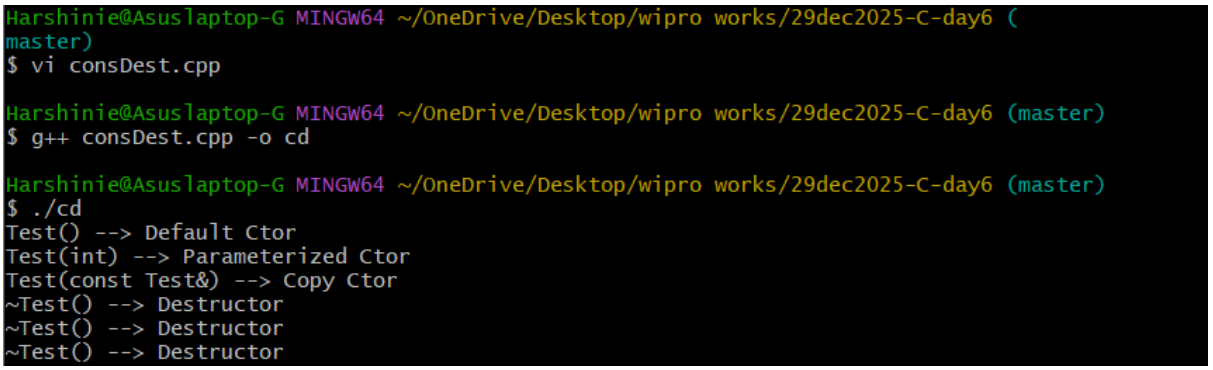
C/C++ PROGRAMMING - HARSHINIE M - DAY 6

1.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate the working of different types of
 * constructors and a destructor in a class.
 * The program should illustrate:
 * 1. Default Constructor
 * 2. Parameterized Constructor
 * 3. Copy Constructor
 * 4. Destructor call sequence
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Test {
public:
    // Default Constructor
    Test() {
        cout << "Test() --> Default Ctor" << endl;
    }

    // Parameterized Constructor
    Test(int) {
        cout << "Test(int) --> Parameterized Ctor" << endl;
    }
```

```
    // Copy Constructor
    Test(const Test &) {
        cout << "Test(const Test&) --> Copy Ctor" << endl;
    }

    // Destructor
    ~Test() {
        cout << "~Test() --> Destructor" << endl;
    }
};

int main() {
    Test objOne;           // Calls Default Constructor
    Test objTwo = 100;     // Calls Parameterized Constructor
    Test objThree = objTwo; // Calls Copy Constructor
}
```



2.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate the invocation of the copy constructor
 * when an object is passed to a function using call by value.
 * The program also shows the order of constructor and destructor calls.
 *
 * Author: Harshinie M
```

```
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Test {
public:
    // Default Constructor
    Test() {
        cout << "Test() --> Default Ctor" << endl;
    }

    // Parameterized Constructor
    Test(int) {
        cout << "Test(int) --> Parameterized Ctor" << endl;
    }

    // Copy Constructor
    Test(const Test &) {
        cout << "Test(const Test&) --> Copy Ctor" << endl;
    }

    // Destructor
    ~Test() {
        cout << "~Test() --> Destructor" << endl;
    }
};

// Function demonstrating call by value
Test fun(Test arg) {
    return arg;
}

int main() {
    Test objTwo = 100; // Calls Parameterized Constructor
    fun(objTwo);      // Copy Constructor is invoked
```

}

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi copyCallbyValue.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ copyCallbyValue.cpp -o value

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./value
Test(int) --> Parameterized Ctor
Test(const Test&) --> Copy Ctor
Test(const Test&) --> Copy Ctor
~Test() --> Destructor
~Test() --> Destructor
~Test() --> Destructor
```

3.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate passing an object to a function
 * using call by reference and to show that the copy constructor
 * is NOT invoked in this case.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Test {
public:
    // Default Constructor
    Test() {
        cout << "Test() --> Default Ctor" << endl;
    }

    // Parameterized Constructor
    Test(int) {
        cout << "Test(int) --> Parameterized Ctor" << endl;
```

```
    }

    // Copy Constructor
    Test(const Test &) {
        cout << "Test(const Test&) --> Copy Ctor" << endl;
    }

    // Destructor
    ~Test() {
        cout << "~Test() --> Destructor" << endl;
    }
};

// Function demonstrating call by reference
Test& fun(Test& arg) {
    return arg;
}

int main() {
    Test objTwo = 100; // Calls Parameterized Constructor
    fun(objTwo);      // No Copy Constructor call
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi copyCallrefer.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ copyCallrefer.cpp -o ref

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./ref
Test(int) --> Parameterized Ctor
~Test() --> Destructor
```

4.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate the use of a friend function
 * to access private members of a class. The program shows that
```

```
 * the main() function can access a class's private constructor
 * and private member functions when declared as a friend.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Test { // All members of a class are private by default
    int data;

    // Private constructor with default argument
    Test(int x = 10) : data(x) {
        cout << "constructor called" << endl;
    }

    // Private member function
    void disp() {
        cout << "Data: " << data << endl;
    }

    // Granting access to main()
    friend int main();
};

int main() {
    Test obj = 100; // Accessing private constructor
    obj.disp();     // Accessing private member function
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi friendmem1.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ friendmem1.cpp -o mem1

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./mem1
constructor called
Data: 100
```

5.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate the use of a friend function
 * to access private data members of a class.
 * The program shows how a non-member function can access
 * private members when declared as a friend.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Box {
    int length; // private data member

public:
    // Parameterized constructor with default value
    Box(int len = 10) : length(len) {}

    // Friend function declaration
    friend void showLength(Box &b);
};

// Friend function definition
void showLength(Box &b) {
```

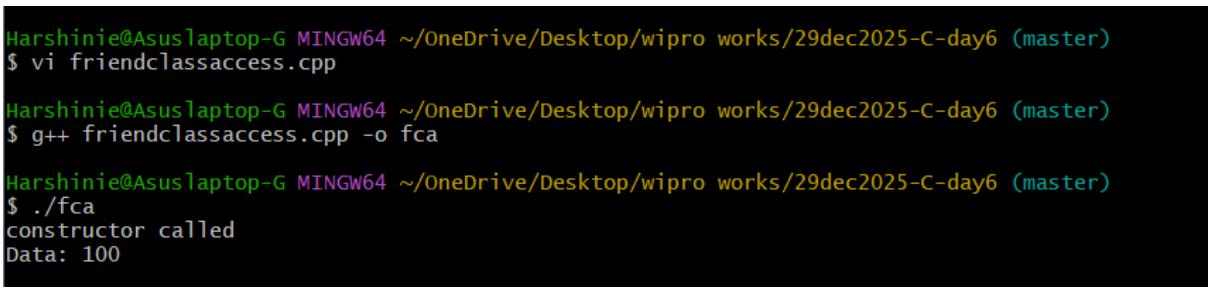
```
    cout << "Length : " << b.length << endl;
}

int main() {
    Box gift;        // Object created using default argument
    showLength(gift); // Friend function accessing private data
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi frienddata1.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ frienddata1.cpp -o data

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./data
Length : 10
```

6.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate the use of a friend class
 * to access private members (private constructor and private
 * member function) of another class.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Test { // All members of a class are private by default
    int data;

    // Private constructor
    Test(int x = 10) : data(x) {
        cout << "constructor called" << endl;
    }
```

```
    // Private member function
    void disp() {
        cout << "Data: " << data << endl;
    }

    // Declaring friend class
    friend class UsingTest;
};

class UsingTest {
    Test obj;

public:
    // Constructor accessing Test's private constructor
    UsingTest(int x = 100) : obj(Test(x)) {}

    // Member function accessing Test's private function
    void print() {
        obj.disp();
    }
};

int main() {
    UsingTest obj = 100;
    obj.print();
}
```



7.

```
/*
 * Purpose:
 * Write a C++ program to demonstrate single inheritance
 * with public inheritance. The program shows that public
 * member functions of a base class can be accessed using
 * a derived class object.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Base {
public:
    void funOne() {
        cout << "Base::funOne()" << endl;
    }

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

// Derived class publicly inheriting Base
class Derived : public Base {};

int main() {
    Derived dObj;   // Derived class object
    dObj.funOne();  // Accessing Base class function
    dObj.funTwo();  // Accessing Base class function
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi singleinherit.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ singleinherit.cpp -o inheritsing

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./inheritsing
Base::funOne()
Base::funTwo()
```

8.

```
/*
 * Purpose:
 * Demonstrate class abstraction, constructors, and copy constructor
 * using an array-based class in C++.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class MyArray { // abstraction
    int arr[100];
    int size;

public:
    MyArray();
    MyArray(int data);
    MyArray(const MyArray&);
    void printArray();
    MyArray& extend(MyArray&);
};

int main() {
```

```
    MyArray objOne;
    MyArray objTwo(100);
    MyArray objThree = objTwo;

    objOne.printArray();
    objTwo.printArray();
    objThree.printArray();

    // objFour = objTwo.extend(objThree);
    // objFour.printArray();
}

MyArray::MyArray() {
    size = 100;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = 0;
}

MyArray::MyArray(int data) {
    size = 100;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = data + cnt;
}

MyArray::MyArray(const MyArray& rhsObj) {
    size = rhsObj.size;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = rhsObj.arr[cnt];
}

void MyArray::printArray() {
    cout << "Size: " << size << "\tarray:" << endl;
    for (int cnt = 0; cnt < size; cnt++)
        cout << arr[cnt] << "  ";
    cout << endl;
}
```

```
MyArray& MyArray::extend(MyArray& rhsObject) {
    // Future implementation using dynamic memory
    return rhsObject;
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi arraycons1.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ arraycons1.cpp -o cons1

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./cons1
Size: 100       array:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
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  1
41  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  18
2  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  1
41  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  18
2  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
```

9.

```
/*
 * Purpose:
 * Demonstrate array abstraction, constructors, copy constructor,
 * and object extension using a class in C++.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class MyArray { // abstraction
    static const int max = 100;
    int arr[max];
```

```
    int size;

public:
    MyArray();                // size = 10
    MyArray(int data);        // size = 20
    MyArray(const MyArray&);  // copy constructor
    void printArray();
    MyArray extend(MyArray&);  // extend two arrays
};

int main() {
    MyArray objOne;
    MyArray objTwo(100);
    MyArray objThree = objTwo; // copy constructor called

    objOne.printArray();
    objTwo.printArray();
    objThree.printArray();

    MyArray objFour = objOne.extend(objThree);
    objFour.printArray();
}

MyArray::MyArray() {
    size = 10;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = 0;
}

MyArray::MyArray(int data) {
    size = 20;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = data + cnt;
}

MyArray::MyArray(const MyArray& rhsObj) {
    size = rhsObj.size;
```

```
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = rhsObj.arr[cnt];
}

void MyArray::printArray() {
    cout << "Size: " << size << "\tarray:" << endl;
    for (int cnt = 0; cnt < size; cnt++)
        cout << arr[cnt] << "  ";
    cout << endl;
}

MyArray MyArray::extend(MyArray& rhsObj) {
    MyArray newObj;
    newObj.size = size + rhsObj.size;

    for (int cnt = 0; cnt < size; cnt++)
        newObj.arr[cnt] = arr[cnt];

    for (int nCnt = 0, cnt = size; cnt < newObj.size; cnt++)
        newObj.arr[cnt] = rhsObj.arr[nCnt++];

    return newObj;
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi arrayextend.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ arrayextend.cpp -o exte

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./exte
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  0  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114
  115  116  117  118  119
```

10.

```
/*
 * Purpose:
```

```
 * Demonstrate object copying in a class using an array in C++.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class MyArray { // abstraction
    int arr[100]; // array
    int size;     // integer

public:
    MyArray();
    MyArray(int data);
    void printArray();
};

int main() {
    MyArray objTwo(100);       // parameterized constructor
    MyArray objThree = objTwo; // object copying

    objTwo.printArray();
    objThree.printArray();
}

MyArray::MyArray() {
    size = 100;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = 0;
}

MyArray::MyArray(int data) {
    size = 100;
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = data + cnt;
```

```
}

void MyArray::printArray() {
    cout << "Size: " << size << "\tarray:" << endl;
    for (int cnt = 0; cnt < size; cnt++)
        cout << arr[cnt] << "  ";
    cout << endl;
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi objectcopyarray.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ objectcopyarray.cpp -o oca

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./oca
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  1
41  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  18
2  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  1
41  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  18
2  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
```

11.

```
/*
 * Purpose:
 * Demonstrate deep copy using copy constructor
 * to avoid dangling pointer issues in C++.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

// Code demonstrates deep copy to avoid dangling pointer
class MyArray {
    int *arr;
```

```
    int size;

public:
    MyArray();
    MyArray(int data);
    MyArray(const MyArray&);
    void printArray();

    ~MyArray() {
        if (size) {
            delete[] arr;
        }
        size = 0;
    }
};

int main() {
    MyArray objTwo(100); // parameterized constructor
    objTwo.printArray();

    {
        MyArray objThree = objTwo; // copy constructor (deep copy)
        objThree.printArray();
    } // objThree destroyed here

    objTwo.printArray(); // safe access, no dangling pointer
}

MyArray::MyArray(const MyArray& rhs) {
    size = rhs.size;
    arr = new int[size]; // separate memory allocation
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = rhs.arr[cnt];
}

MyArray::MyArray() : size(10) {
    arr = new int[size];
```

```
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = 0;
}

MyArray::MyArray(int sz) : size(sz) {
    int data = 101;
    arr = new int[size];
    for (int cnt = 0; cnt < size; cnt++)
        arr[cnt] = data + cnt;
}

void MyArray::printArray() {
    cout << "Size: " << size << "\tarray:" << endl;
    for (int cnt = 0; cnt < size; cnt++)
        cout << arr[cnt] << "  ";
    cout << endl;
}
```

```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ vi deepcopyarray.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ g++ deepcopyarray.cpp -o dca

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/29dec2025-C-day6 (master)
$ ./dca
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  1
42  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  18
3  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  1
42  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  18
3  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  1
42  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  18
3  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200
```

12.

/*
*** Purpose:**

```
 * Demonstrate static data member and static member function in
C++.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Employee {
    static string compName; // static data member

public:
    static string company() { // static member function
        return compName;
    }
};

int main() {
    cout << "Employee::company() --> " << Employee::company() <<
endl;

    Employee obj1;
    cout << "obj1.company() --> " << obj1.company() << endl;
}

// Static data member definition
string Employee::compName = "Wipro Ltd";
```

13.

```
/*
 * Purpose:
 * Demonstrate Singleton design pattern in C++ using static member
 * to ensure only one object is created.
 *
 * Author: Harshinie M
 * Date  : 29-12-2025
 */

#include <iostream>
using namespace std;

class Singleton {
private:
    static Singleton *ptr2Obj;

    // Private constructors
    Singleton() {
        cout << "Object created..." << endl;
    }
    Singleton(int x) {
        cout << "Object created...with " << x << endl;
    }
    Singleton(const Singleton &);

public:
```

```
    // Static method to get single instance
    static Singleton& getSingleton() {
        if (ptr2Obj == nullptr)
            ptr2Obj = new Singleton();
        return *ptr2Obj;
    }

    void disp() {
        cout << "Singleton::disp()" << endl;
    }
};

// Initialize static member
Singleton* Singleton::ptr2Obj = nullptr;

int main() {
    // Singleton objOne; // Error: constructor is private

    Singleton &refObj = Singleton::getSingleton();
    refObj.disp();
}
```