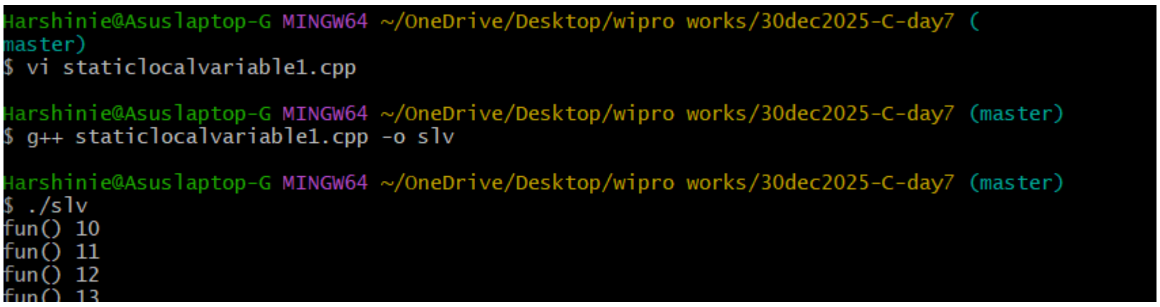
C/c++ Documentation - Day 7 - Harshinie M

1.

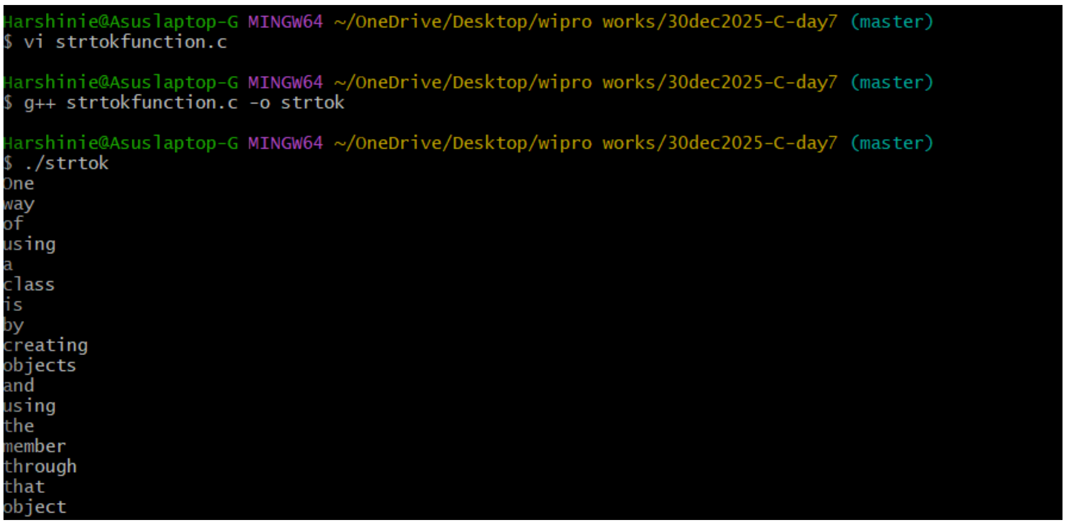
```
/*
 * Purpose : Write a program to demonstrate how a static local variable
 *          inside a function retains its value across multiple
 *          function calls.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

// Function declaration
int fun();

int main(){
    // Calling fun() multiple times
    cout << "fun() " << fun() << endl;
    cout << "fun() " << fun() << endl;
    cout << "fun() " << fun() << endl;
    cout << "fun() " << fun() << endl;

    return 0;
}

// Function definition
int fun(){
    // Static local variable
    // Initialized only once and value is retained between function calls
    static int var = 10;

    // Return current value and then increment
    return var++;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (
master)
$ vi staticlocalvariable1.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ staticlocalvariable1.cpp -o slv

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./slv
fun() 10
fun() 11
fun() 12
fun() 13
```

2.

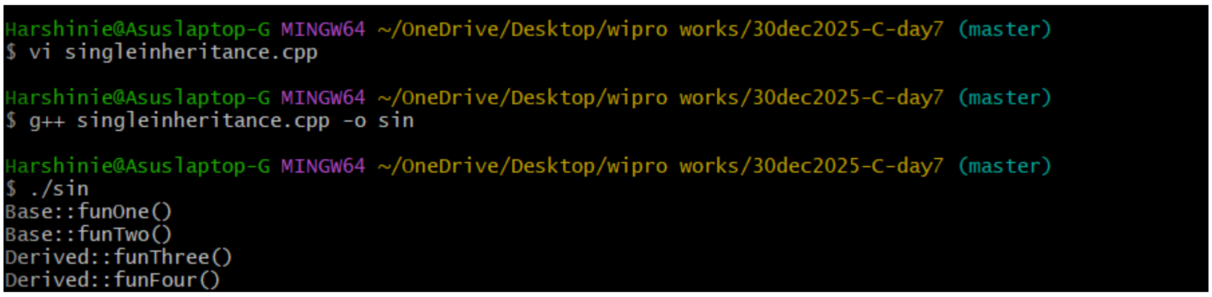
```
/*
 * Purpose : Write a program to demonstrate how the strtok() function
 *           is used to split a string into tokens using space as
 *           a delimiter.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <stdio.h>
#include <string.h>

int main(){
    // Input string to be tokenized
    char str[] =
        "One way of using a class is by creating objects and using the member through that
object";

    // First call to strtok(): pass string address and delimiter
    char *ptr = strtok(str, " ");

    // Loop until no more tokens are found
    while(ptr != NULL){
        // Print each token
        printf("%s\n", ptr);

        // Subsequent calls: pass NULL to continue tokenizing same string
        ptr = strtok(NULL, " ");
    }

    return 0;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi strtokfunction.c

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ strtokfunction.c -o strtok

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./strtok
One
way
of
using
a
class
is
by
creating
objects
and
using
the
member
through
that
object
```

3.

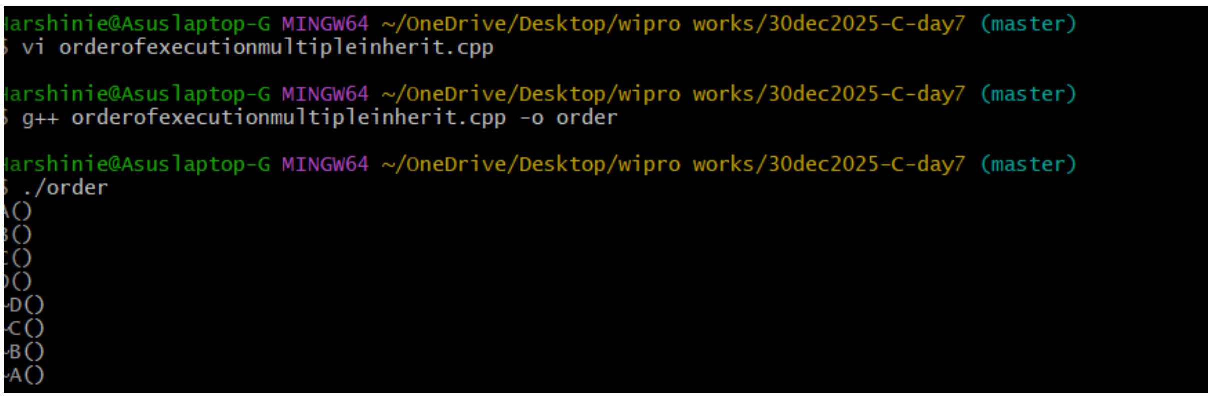
```
/*
 * Purpose : Write a program to demonstrate single inheritance in C++,
 *          where a derived class accesses public member functions
 *          of a base class.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

// Base class
class Base{
public:
    // Member function of Base class
    void funOne(){
        cout << "Base::funOne()" << endl;
    }

    // Another member function of Base class
    void funTwo(){
        cout << "Base::funTwo()" << endl;
    }
};

// Derived class inherits from Base
class Derived : public Base{
public:
    // Member function of Derived class
    void funThree(){
        cout << "Derived::funThree()" << endl;
    }

    // Another member function of Derived class
    void funFour(){
        cout << "Derived::funFour()" << endl;
    }
};

int main(){
    // Create object of Derived class
    Derived dObj;

    // Access Base class functions using Derived object
    dObj.funOne();
    dObj.funTwo();
    // Access Derived class functions
    dObj.funThree();
    dObj.funFour();

    return 0;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi singleinheritance.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ singleinheritance.cpp -o sin

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./sin
Base::funOne()
Base::funTwo()
Derived::funThree()
Derived::funFour()
```

4.

```
/*
 * Purpose : Write a program to demonstrate the order of execution of
 *           constructors and destructors in multiple inheritance.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

// Base class A
struct A{
    A(){
        cout << "A()" << endl;
    }
    ~A(){
        cout << "~A()" << endl;
    }
};

// Base class B
struct B{
    B(){
        cout << "B()" << endl;
    }
    ~B(){
        cout << "~B()" << endl;
    }
};

// Base class C
struct C{
    C(){
        cout << "C()" << endl;
    }
    ~C(){
        cout << "~C()" << endl;
    }
};

// Derived class D inherits from A, B, and C
struct D : A, B, C{
    // Constructor
    D() : B(), C(), A(){
        cout << "D()" << endl;
    }

    // Destructor
    ~D(){
        cout << "~D()" << endl;
    }
};

int main(){
    // Creating object of derived class D
    D dObj;

    // Destructors will be called automatically
    return 0;
}
```

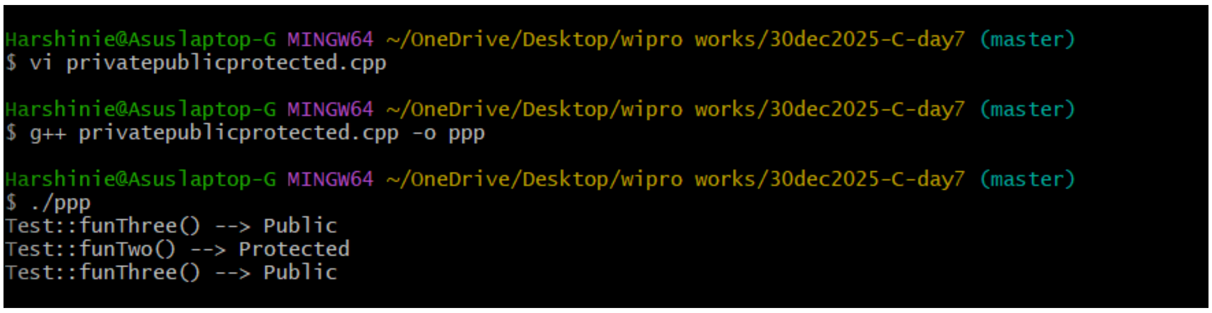


```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi orderofexecutionmultipleinherit.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ orderofexecutionmultipleinherit.cpp -o order

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./order
A()
B()
C()
D()
~D()
~C()
~B()
~A()
```

5.

```
/*
 * Purpose : Write a program to demonstrate how private, protected,
 *          and public members of a base class are accessed in
 *          a derived class using public inheritance.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Test{
   // Private member by default
   void funOne(){
       cout << "Test::funOne() --> Private" << endl;
   }

protected:
   // Protected member
   void funTwo(){
       cout << "Test::funTwo() --> Protected" << endl;
   }

public:
   // Public member
   void funThree(){
       cout << "Test::funThree() --> Public" << endl;
   }
};

// Derived class using public inheritance
class Derived : public Test{
public:
   void funFour(){
       // funOne();  //  Not accessible (private member)

       funTwo();    //  (protected member)
       funThree();  // (public member)
   }
};

int main(){
   // Create object of Derived class
   Derived dObj;

   // Public member can be accessed using object
   dObj.funThree();
```

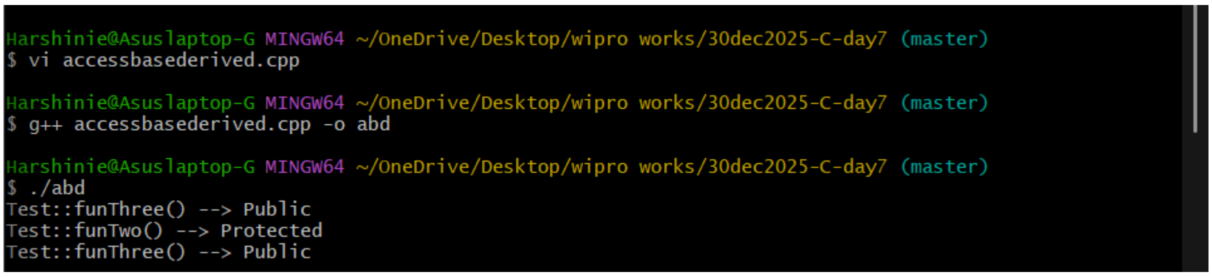
```
    // Function of Derived class accessing inherited members
    dObj.funFour();

    return 0;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi privatepublicprotected.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ privatepublicprotected.cpp -o ppp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./ppp
Test::funThree() --> Public
Test::funTwo() --> Protected
Test::funThree() --> Public
```

6.

```
/*
 * Purpose : Write a program to demonstrate the accessibility of
 *           private, protected, and public members of a base class
 *           in a derived class using public inheritance.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Test{
    // Private member by default
    void funOne(){
        cout << "Test::funOne() --> Private" << endl;
    }

protected:
    // Protected member
    void funTwo(){
        cout << "Test::funTwo() --> Protected" << endl;
    }

public:
    // Public member
    void funThree(){
        cout << "Test::funThree() --> Public" << endl;
    }
};

// Derived class with public inheritance
class Derived : public Test{
public:
    void funFour(){
        // funOne();   //  Not accessible (private member)

        funTwo();     //  Accessible (protected member)
        funThree();   //  Accessible (public member)
    }
};

int main(){
    Derived dObj;

    // Accessing public member using object
    dObj.funThree();

    // Accessing inherited members inside derived class function
    dObj.funFour();

    return 0;
}
```

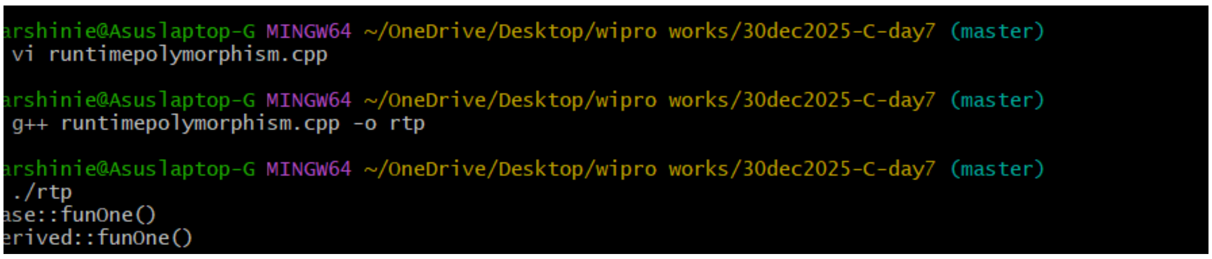


```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi accessbasederived.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ accessbasederived.cpp -o abd

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./abd
Test::funThree() --> Public
Test::funTwo() --> Protected
Test::funThree() --> Public
```

7.

```
/*
 * Purpose : Write a program to demonstrate runtime polymorphism
 *           using virtual functions and base class pointers.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

// Base class
```

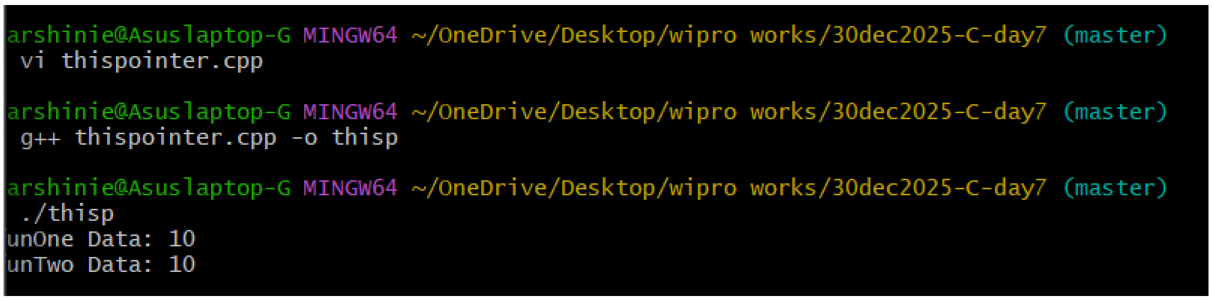
```
class Base{
public:
    // Virtual function
    virtual void funOne(){
        cout << "Base::funOne()" << endl;
    }
};

// Derived class
class Derived : public Base{
public:
    // Overridden function
    void funOne(){
        cout << "Derived::funOne()" << endl;
    }
};

int main(){
    // Base class pointer and objects
    Base *ptr, bObj;
    Derived dObj;

    // Base pointer pointing to Base object
    ptr = &bObj;
    ptr->funOne();   // Calls Base::funOne()

    // Base pointer pointing to Derived object
    ptr = &dObj;
    ptr->funOne();   // Calls Derived::funOne() (runtime binding)

    return 0;
}
```



```
arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 vi runtimepolymorphism.cpp

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 g++ runtimepolymorphism.cpp -o rtp

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 ./rtp
ase::funOne()
erived::funOne()
```

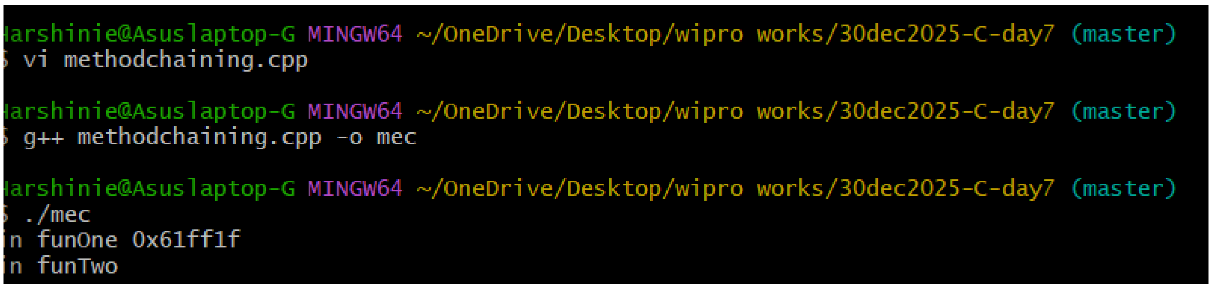
8.

```
/*
 * Purpose : Write a program to demonstrate the use of the
 *          this pointer to differentiate between data members
 *          and function parameters having the same name.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Test{
    int data;   // Data member of the class

public:
    // Function with parameter name same as data member
    void funOne(int data){
        // 'this->data' refers to the data member
        // 'data' refers to the function parameter
        this->data = data;
        cout << "funOne Data: " << data << endl;
    }

    // Function to display the stored data
    void funTwo(){
        cout << "funTwo Data: " << data << endl;
    }
};

int main(){
    // Create object of Test class
    Test obj;

    // Assign value using funOne()
    obj.funOne(10);

    // Display value using funTwo()
    obj.funTwo();

    return 0;
}
```



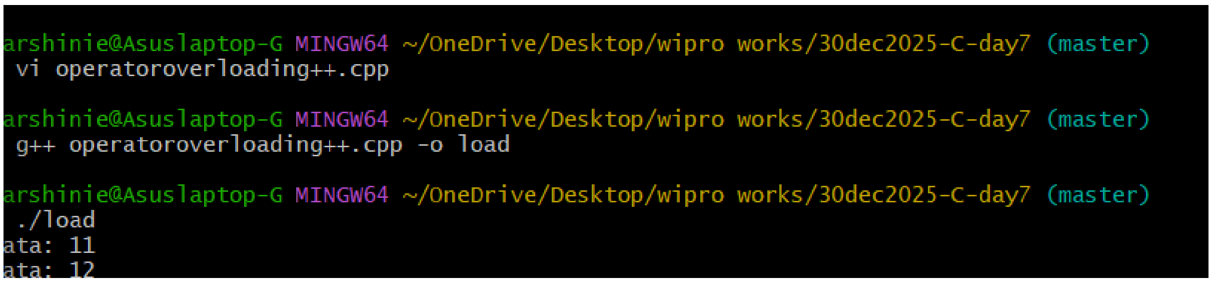
```
arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 vi thispointer.cpp

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 g++ thispointer.cpp -o thisp

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 ./thisp
unOne Data: 10
unTwo Data: 10
```

```
9.
/*
 * Purpose : Write a program to demonstrate returning the current
 *           object using the this pointer and perform method chaining.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Test{
public:
    // Function returns reference to current object
    Test& funOne(){
        cout << "in funOne " << this << endl;
        return *this;   // return current object
    }

    // Another member function
    void funTwo(){
        cout << "in funTwo" << endl;
    }
};

int main(){
    // Create object of Test class
    Test objOne;

    // Method chaining using returned object
    objOne.funOne().funTwo();

    return 0;
}
```



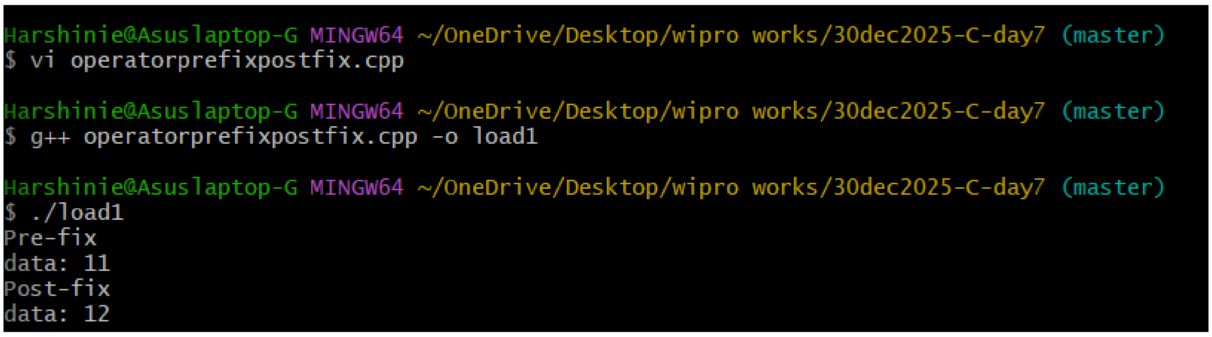
```
larshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 vi methodchaining.cpp

larshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 g++ methodchaining.cpp -o mec

larshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 ./mec
n funOne 0x61ff1f
n funTwo
```

```
10.

/*
 * Purpose : Write a program to demonstrate operator overloading
 *          by overloading the prefix increment (++) operator
 *          using member function and returning object reference.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Num{
   int data;   // Data member

public:
   // Constructor with default argument
   Num(int x = 0) : data(x) {}

   // Overloading prefix increment operator
   Num& operator++(){
      data++;          // Increment data
      return *this;    // Return current object
   }

   // Display function
   void disp(ostream &out){
      out << "data: " << data << endl;
   }
};

int main(){
   // Create object and initialize
   Num a = 10;

   // Prefix increment
   ++a;
   a.disp(cout);

   // Prefix increment again
   ++a;
   a.disp(cout);

   return 0;
}
```


```
arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 vi operatoroverloading++.cpp

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 g++ operatoroverloading++.cpp -o load

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 ./load
ata: 11
ata: 12
```

11.

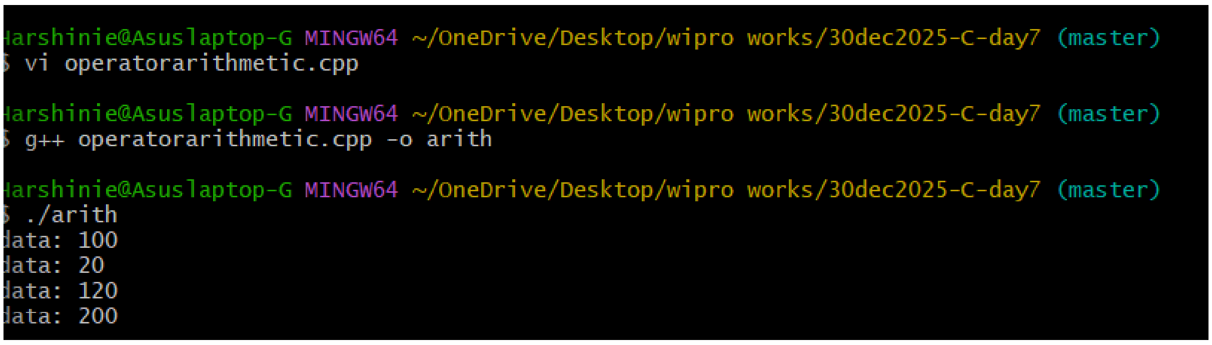
```
/*
 * Purpose : Write a program to demonstrate operator overloading
 *           by overloading both prefix (++) and postfix (++)
 *           increment operators and show their behavior.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Num{
    int data;   // Data member

public:
    // Constructor with default value
    Num(int x = 0) : data(x) {}

    // Prefix increment operator overloading
    Num& operator++(){
        cout << "Pre-fix" << endl;
        data++;              // Increment first
        return *this;        // Return modified object
    }

    // Postfix increment operator overloading
    Num operator++(int){
        cout << "Post-fix" << endl;

        // Store current state before increment
        Num temp(*this);

        data++;              // Increment after storing
        return temp;          // Return old value
    }
    // Display function
    void disp(ostream &out){
        out << "data: " << data << endl;
    }
};

int main(){
    Num a = 10;

    // Prefix increment
    ++a;
    a.disp(cout);

    // Postfix increment
    a++;
    a.disp(cout);

    return 0;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi operatorprefixpostfix.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ operatorprefixpostfix.cpp -o load1

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./load1
Pre-fix
data: 11
Post-fix
data: 12
```

12.

```
/*
 * Purpose : Write a program to demonstrate operator overloading
 *           for arithmetic operators (+ and -) using member functions.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Num{
    int data;   // Data member

public:
    // Constructor with default value
    Num(int x = 0) : data(x) {}

    // Overload '+' operator
    Num operator+(const Num& rhs){
        Num temp(data + rhs.data);  // Add data members
        return temp;                // Return new object
    }

    // Overload '-' operator
    Num operator-(const Num& rhs){
        Num temp(data - rhs.data);  // Subtract data members
        return temp;                // Return new object
    }

    // Display function
    void disp(ostream &out){
        out << "data: " << data << endl;
    }
};

int main(){
    // Create objects
    Num a = 100, b = 20;

    // Display initial values
    a.disp(cout);
    b.disp(cout);

    // Use overloaded '+' operator
    Num c = a + b;

    // Use combination of '+' and '-' operators
    Num d = c + a - b;

    // Display results
    c.disp(cout);
    d.disp(cout);

    return 0;
}
```

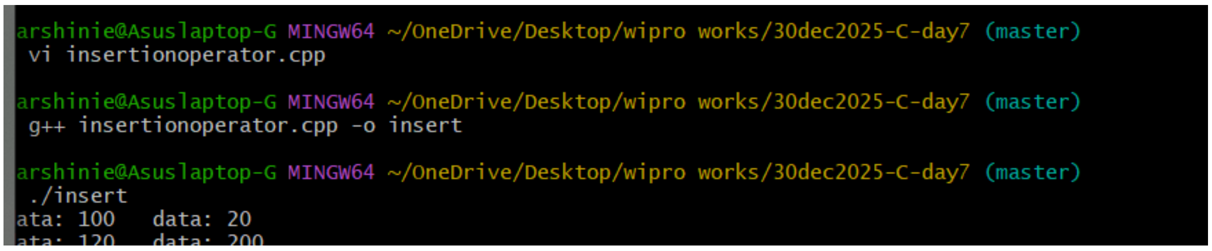


```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi operatorarithmetic.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ operatorarithmetic.cpp -o arith

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./arith
data: 100
data: 20
data: 120
data: 200
```

13.

```
/*
 * Purpose : Write a program to demonstrate operator overloading
 *           for arithmetic operators (+ and -) as member functions
 *           and overloading the insertion operator (<<) using
 *           a friend function.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Num{
    int data;   // Data member

public:
    // Constructor with default value
    Num(int x = 0) : data(x) {}

    // Overload '+' operator as member function
    Num operator+(const Num& rhs){
        Num temp(data + rhs.data);  // Add data members
        return temp;
    }

    // Overload '-' operator as member function
    Num operator-(const Num& rhs){
        Num temp(data - rhs.data);  // Subtract data members
        return temp;
    }

    // Friend function to overload '<<' operator
    friend ostream& operator<<(ostream &out, const Num &obj);
};
```

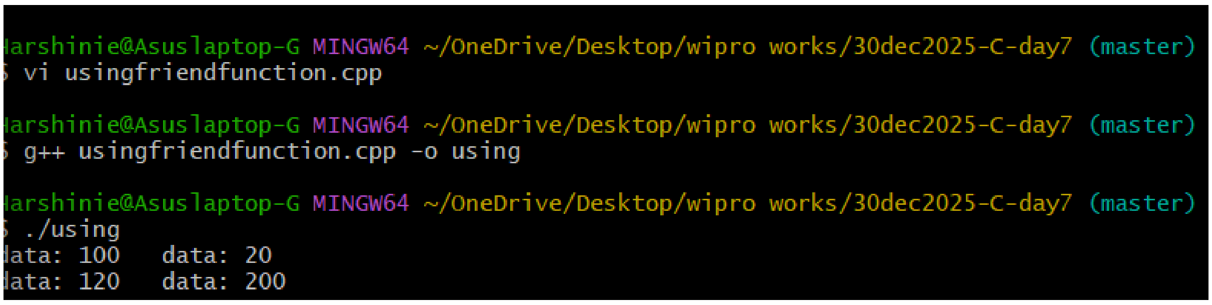
```
// Overloading '<<' operator using friend function
ostream& operator<<(ostream &out, const Num &obj){
    out << "data: " << obj.data;
    return out;  // Allow chaining
}

int main(){
    // Create objects
    Num a = 100, b = 20;

    // Display using overloaded '<<'
    cout << a << "   " << b << endl;

    // Perform arithmetic using overloaded operators
    Num c = a + b;
    Num d = c + a - b;

    // Display results using overloaded '<<'
    cout << c << "   " << d << endl;

    return 0;
}
```



```
arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 vi insertionoperator.cpp

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 g++ insertionoperator.cpp -o insert

arshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
 ./insert
ata: 100   data: 20
ata: 120   data: 200
```

14.

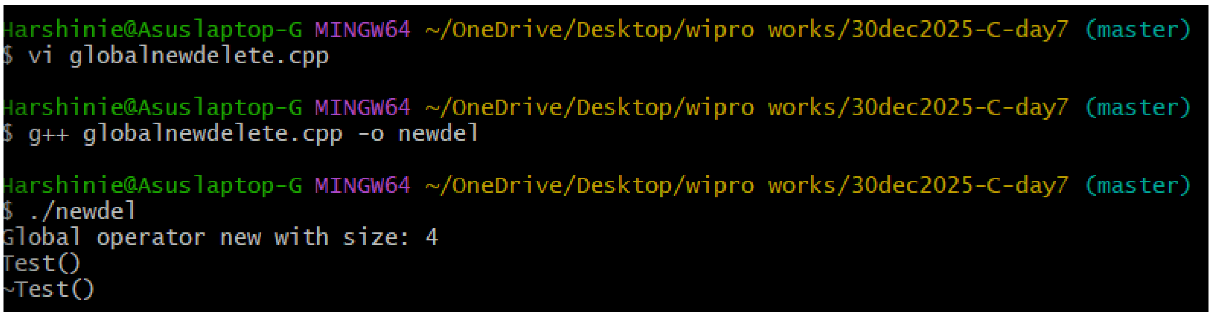
```
/*
 * Purpose : Write a program to demonstrate operator overloading
 *           entirely using friend functions for arithmetic operators
 *           (+, -) and the insertion operator (<<).
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
using namespace std;

class Num{
    int data;   // Data member

public:
    // Constructor with default value
    Num(int x = 0) : data(x) {}

    // Friend functions for operator overloading
    friend Num operator+(const Num&, const Num&);
    friend Num operator-(const Num&, const Num&);
    friend ostream& operator<<(ostream &, const Num &);
};

// Overload '+' operator using friend function
Num operator+(const Num& lhs, const Num& rhs){
    Num temp(lhs.data + rhs.data);
    return temp;
}

// Overload '-' operator using friend function
Num operator-(const Num& lhs, const Num& rhs){
    Num temp(lhs.data - rhs.data);
    return temp;
}

// Overload '<<' operator using friend function
ostream& operator<<(ostream &out, const Num &obj){
    out << "data: " << obj.data;
    return out;  // Allows chaining
}

int main(){
    Num a = 100, b = 20;

    // Display initial objects
    cout << a << "   " << b << endl;

    // Perform arithmetic using overloaded operators
    Num c = a + b;
    Num d = c + a - b;

    // Display results
    cout << c << "   " << d << endl;

    return 0;
}
```



```
larshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi usingfriendfunction.cpp

larshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ usingfriendfunction.cpp -o using

larshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./using
data: 100    data: 20
data: 120    data: 200
```

15.

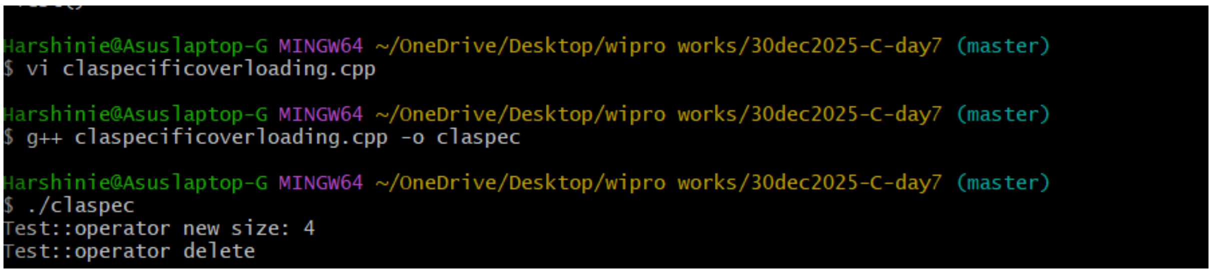
```
/*
 * Purpose : Write a program to demonstrate overloading of global
 *           new and delete operators for dynamic memory allocation
 *           and deallocation.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
#include <cstdlib>  // For malloc and free
using namespace std;

// Overloading global 'new' operator
void* operator new(size_t size){
    cout << "Global operator new with size: " << size << endl;

    void *ptr = malloc(size);  // Allocate memory

    if (ptr == NULL){
        throw bad_alloc();      // Throw exception if allocation fails
    }

    return ptr;                 // Return allocated memory
}

// Overloading global 'delete' operator
void operator delete(void *ptr){
    cout << "Global operator delete" << endl;
    free(ptr);                  // Free allocated memory
}

// Test structure
struct Test{
    int data;

    // Constructor
    Test(int x = 0) : data(x){
        cout << "Test()" << endl;
    }

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

int main(){
    // Dynamic allocation using overloaded global new
    Test *ptr = new Test();

    // Deallocation using overloaded global delete
    delete ptr;

    return 0;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi globalnewdelete.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ globalnewdelete.cpp -o newdel

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./newdel
Global operator new with size: 4
Test()
~Test()
```

16.

```
/*
 * Purpose : Write a program to demonstrate class-specific
 *           overloading of new and delete operators for dynamic
 *           memory allocation and deallocation.
 *
 * Author  : Harshinie M
 * Date    : 30-12-2025
 */

#include <iostream>
#include <cstdlib>  // For malloc and free
using namespace std;

class Test{
    int data;   // Data member
public:
    // Constructor
    Test(int x = 0) : data(x) {}

    // Overloaded class-specific new operator
    void* operator new(size_t size){
        cout << "Test::operator new size: " << size << endl;

        void *ptr = malloc(size);   // Allocate memory
        if (ptr == NULL)
            throw bad_alloc();      // Throw exception if allocation fails
        return ptr;
    }

    // Overloaded class-specific delete operator
    void operator delete(void *ptr){
        cout << "Test::operator delete" << endl;
        free(ptr);                  // Deallocate memory
    }
};

int main(){
    // Dynamic allocation using class-specific new
    Test *ptr = new Test(100);

    // Deallocation using class-specific delete
    delete ptr;

    return 0;
}
```



```
Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ vi claspecificoverloading.cpp

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ g++ claspecificoverloading.cpp -o claspec

Harshinie@Asuslaptop-G MINGW64 ~/OneDrive/Desktop/wipro works/30dec2025-C-day7 (master)
$ ./claspec
Test::operator new size: 4
Test::operator delete
```