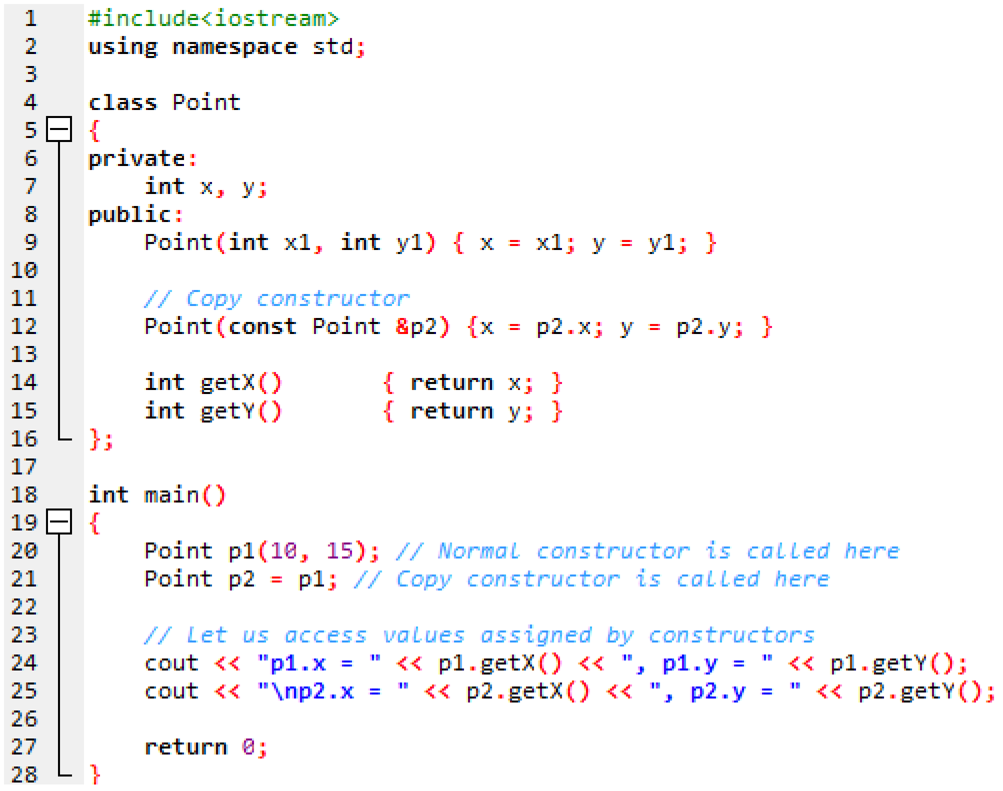
What is a copy constructor?
A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

```
    ClassName (const ClassName &old_obj);
```

Lab 1 Following is a simple example of copy constructor:



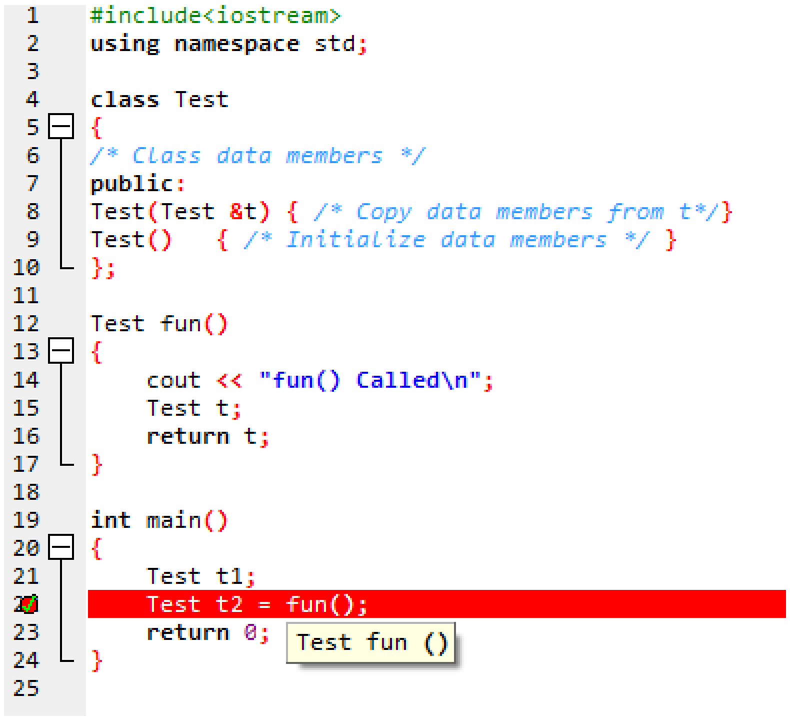
```
#include<iostream>
using namespace std;

class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p2) {x = p2.x; y = p2.y; }

    int getX()       { return x; }
    int getY()       { return y; }
};

int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

    return 0;
}
```

Copy constructor vs Assignment Operator
Which of the following two statements call copy constructor and which one calls assignment operator?

```
MyClass t1, t2;
MyClass t3 = t1;  // ----> (1)
t2 = t1;          // -----> (2)
```

Copy constructor is called when a new object is created from an existing object, as a copy of the existing object. Assignment operator is called when an already initialized object is assigned a new value from another existing object. In the above example (1) calls copy constructor and (2) calls assignment operator.

Lab 2 Why copy constructor argument should be const in C++?



```
#include<iostream>
using namespace std;

class Test
{
/* Class data members */
public:
Test(Test &t) { /* Copy data members from t*/}
Test()    { /* Initialize data members */ }
};

Test fun()
{
    cout << "fun() Called\n";
    Test t;
    return t;
}

int main()
{
    Test t1;
    Test t2 = fun();
    return 0;
}
```

How to fix?

```
Test(const Test &t) { cout << "Copy Constructor Called\n"; }
```

OR

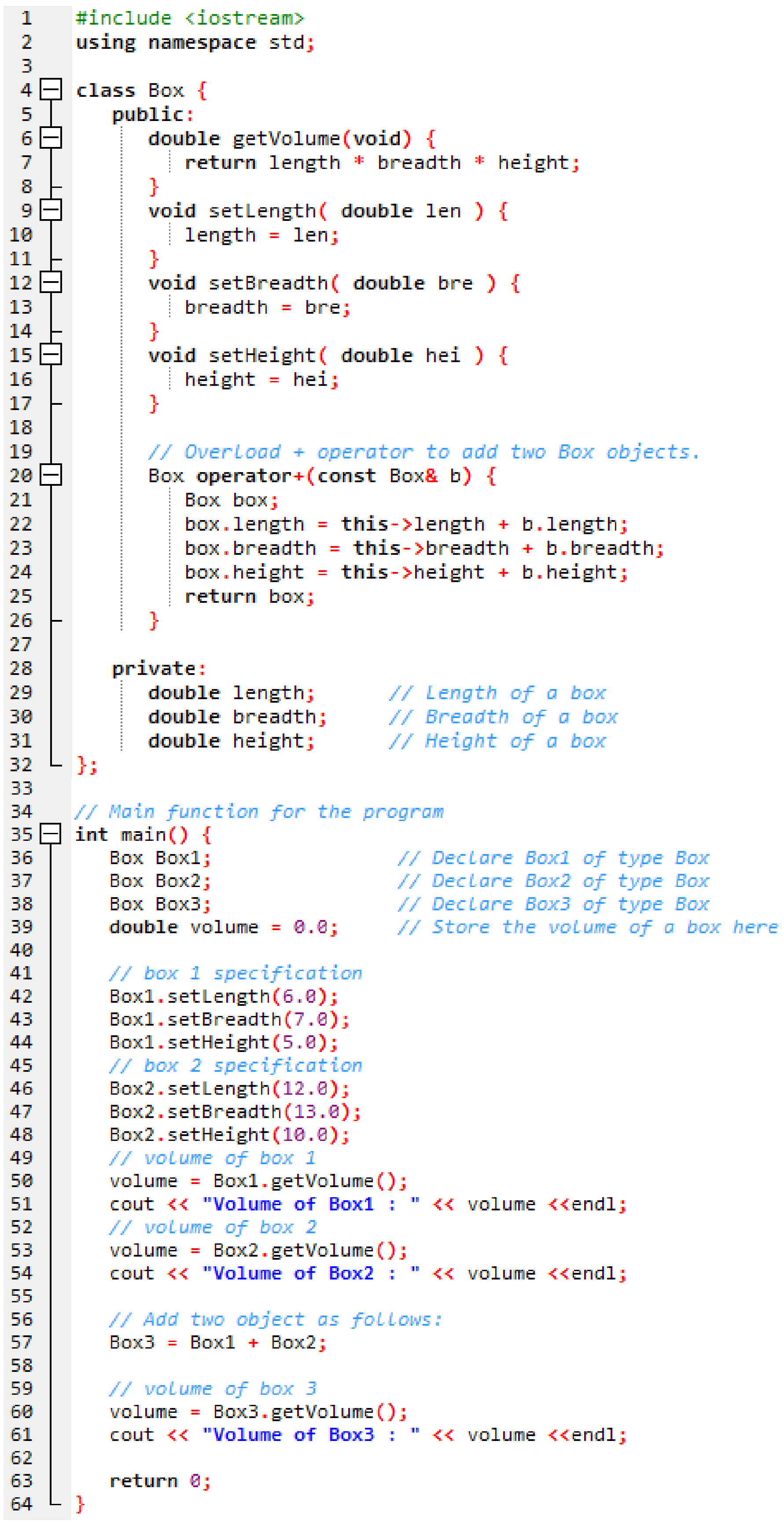
```
Test t2;
t2 = fun();
```

What is Operator Overloading?

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.

Lab 3 : Adding the boxes object value with operator+

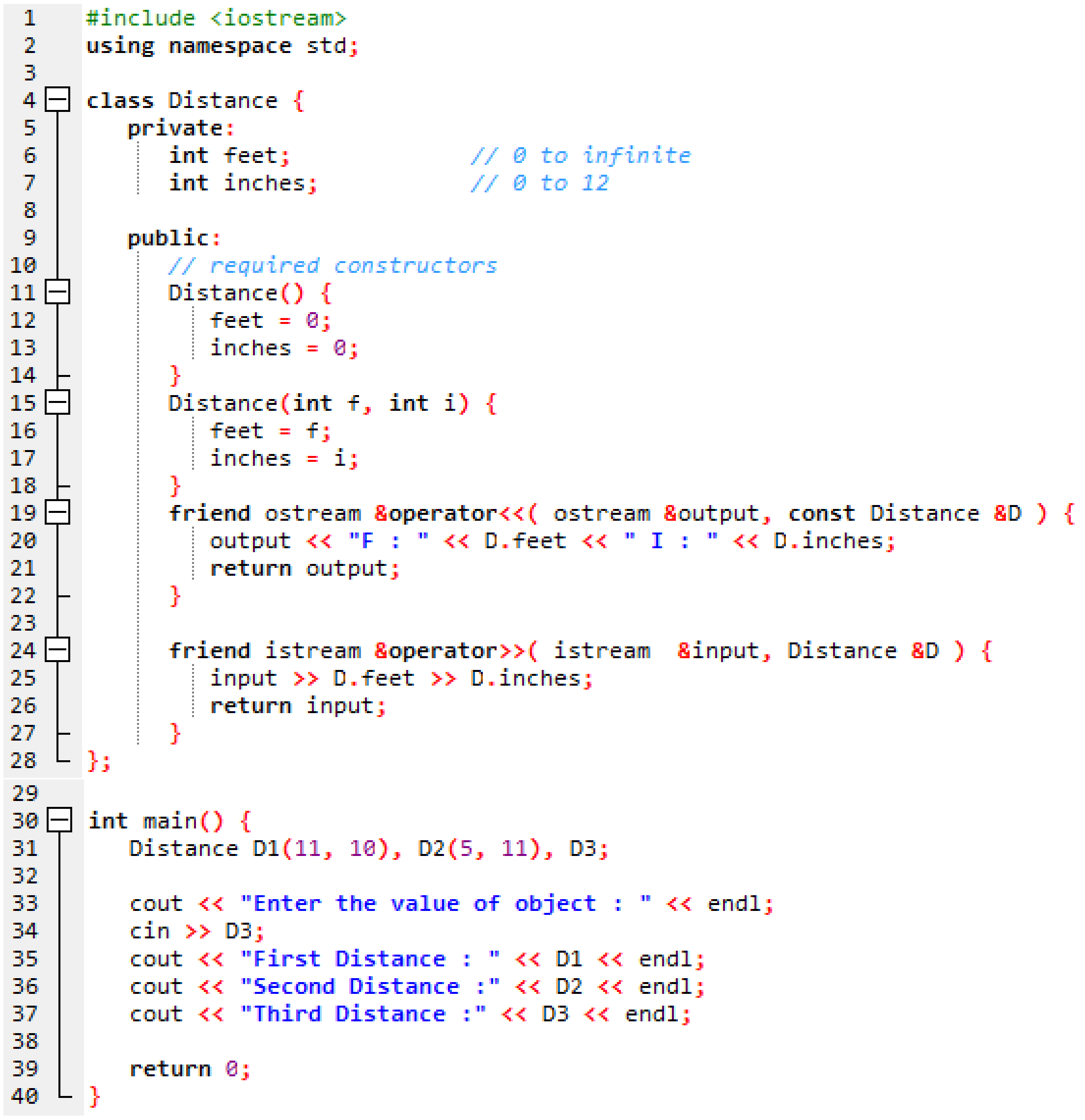


```cpp
#include <iostream>
using namespace std;

class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }
      void setLength( double len ) {
         length = len;
      }
      void setBreadth( double bre ) {
         breadth = bre;
      }
      void setHeight( double hei ) {
         height = hei;
      }

      // Overload + operator to add two Box objects.
      Box operator+(const Box& b) {
         Box box;
         box.length = this->length + b.length;
         box.breadth = this->breadth + b.breadth;
         box.height = this->height + b.height;
         return box;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};

// Main function for the program
int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   Box Box3;                // Declare Box3 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);
   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);
   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;
   // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;

   // Add two object as follows:
   Box3 = Box1 + Box2;

   // volume of box 3
   volume = Box3.getVolume();
   cout << "Volume of Box3 : " << volume <<endl;

   return 0;
}
```

C++ is able to input and output the built-in data types using the stream extraction operator >> and the stream insertion operator <<. The stream insertion and stream extraction operators also can be overloaded to perform input and output for user-defined types like an object.

Here, it is important to make operator overloading function a friend of the class because it would be called without creating an object.

Lab 4 Following example explains how extraction operator >> and insertion operator <<.



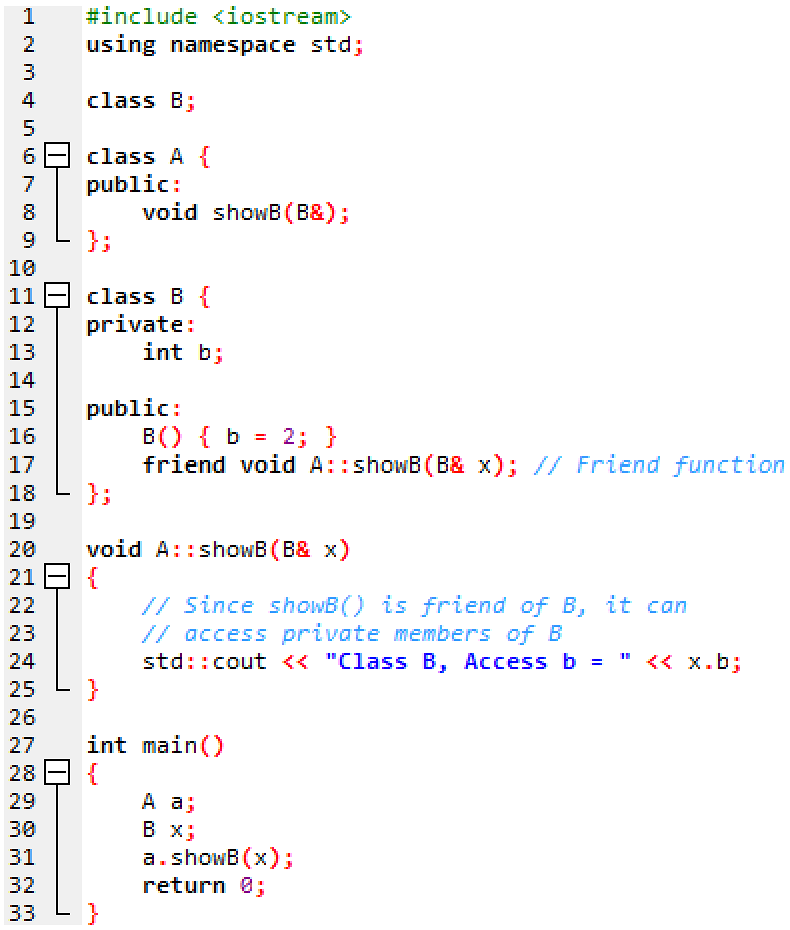
```cpp
#include <iostream>
using namespace std;

class Distance {
   private:
      int feet;             // 0 to infinite
      int inches;           // 0 to 12

   public:
      // required constructors
      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }
      friend ostream &operator<<( ostream &output, const Distance &D ) {
         output << "F : " << D.feet << " I : " << D.inches;
         return output;
      }

      friend istream &operator>>( istream  &input, Distance &D ) {
         input >> D.feet >> D.inches;
         return input;
      }
};

int main() {
   Distance D1(11, 10), D2(5, 11), D3;

   cout << "Enter the value of object : " << endl;
   cin >> D3;
   cout << "First Distance : " << D1 << endl;
   cout << "Second Distance :" << D2 << endl;
   cout << "Third Distance :" << D3 << endl;

   return 0;
}
```

Friend Function Like friend class, a friend function can be given special grant to access private and protected members. A friend function can be:

a) A method of another class

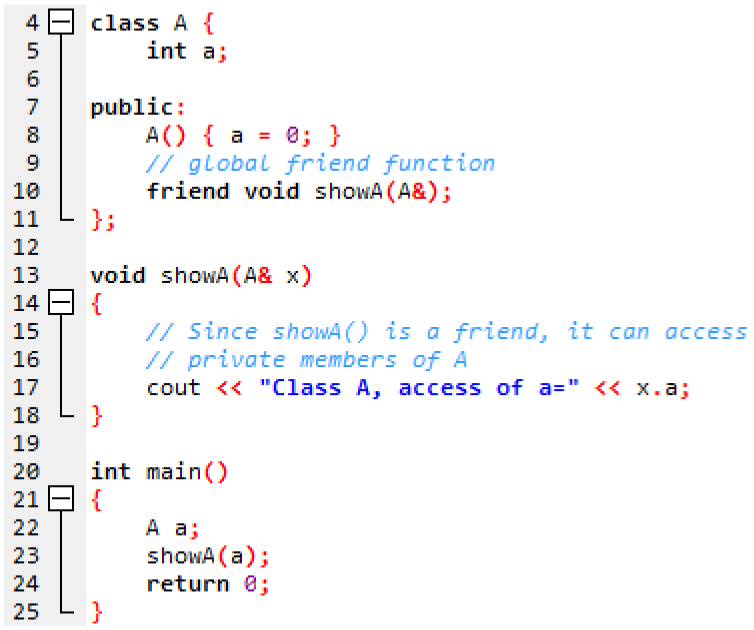
b) A global function

Lab 5   Example (a) - A method of another class



```cpp
#include <iostream>
using namespace std;

class B;

class A {
public:
    void showB(B&);
};

class B {
private:
    int b;

public:
    B() { b = 2; }
    friend void A::showB(B& x); // Friend function
};

void A::showB(B& x)
{
    // Since showB() is friend of B, it can
    // access private members of B
    std::cout << "Class B, Access b = " << x.b;
}

int main()
{
    A a;
    B x;
    a.showB(x);
    return 0;
}
```

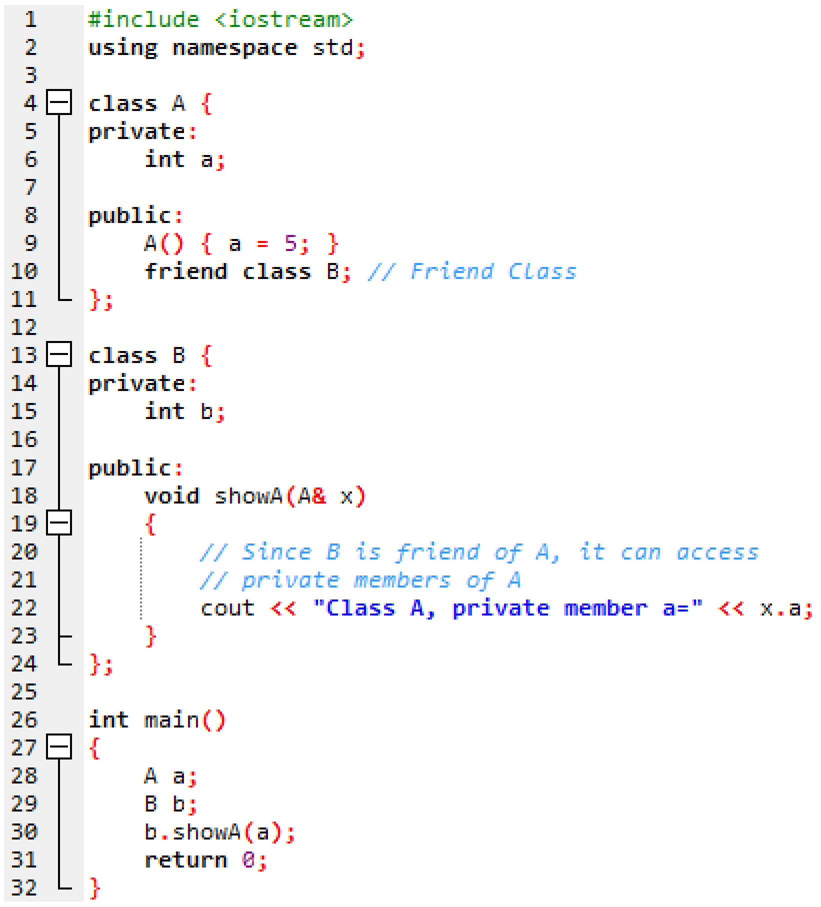
Example (b) - A global function



```cpp
class A {
    int a;

public:
    A() { a = 0; }
    // global friend function
    friend void showA(A&);
};

void showA(A& x)
{
    // Since showA() is a friend, it can access
    // private members of A
    cout << "Class A, access of a=" << x.a;
}

int main()
{
    A a;
    showA(a);
    return 0;
}
```

Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

Lab 6 Example Friend class



```cpp
#include <iostream>
using namespace std;

class A {
private:
    int a;

public:
    A() { a = 5; }
    friend class B; // Friend Class
};

class B {
private:
    int b;

public:
    void showA(A& x)
    {
        // Since B is friend of A, it can access
        // private members of A
        cout << "Class A, private member a=" << x.a;
    }
};

int main()
{
    A a;
    B b;
    b.showA(a);
    return 0;
}
```