SC/CE/CZ2002 Tutorial 9

Agenda

- Common mistakes in C++ programming
- Operator overloading
- Polymorphism

Q1

Q1: Debug and run the following program.

```cpp
#include <iostream.h>

class A {
 protected :
       int a,b;
 public :
        A(int x=0, int y) {
               a = x ;
               b = y ;
        }
        virtual void print() ;


};
class B: public A {
 private:
         float p,q ;
 public :
        B(int m, int n, float u, float v) {
              p = u ;
              q = v ;
        }
        B() { p = q = 0 ; }
        void input(float u, float v) ;
        virtual void print(float) ;
};
void A::print(void) {
       cout << "A values: " << a << " " << b << "\n" ;
}
void B::print(float) {
       cout << "B values : " << u << " " << v << "\n" ;
}
void B::input(float x, float y) {
       p = x ;   q = y ;
}

int doubleIt(A a) { return a.a * a.a ; }

main() {
  A a1(10,20), *ptr ;
  B b1;
  b1.input(7.5, 3.142) ;

  ptr = &a1 ;
  ptr->print() ;
  ptr = &b1;
  ptr->print() ;
}
```

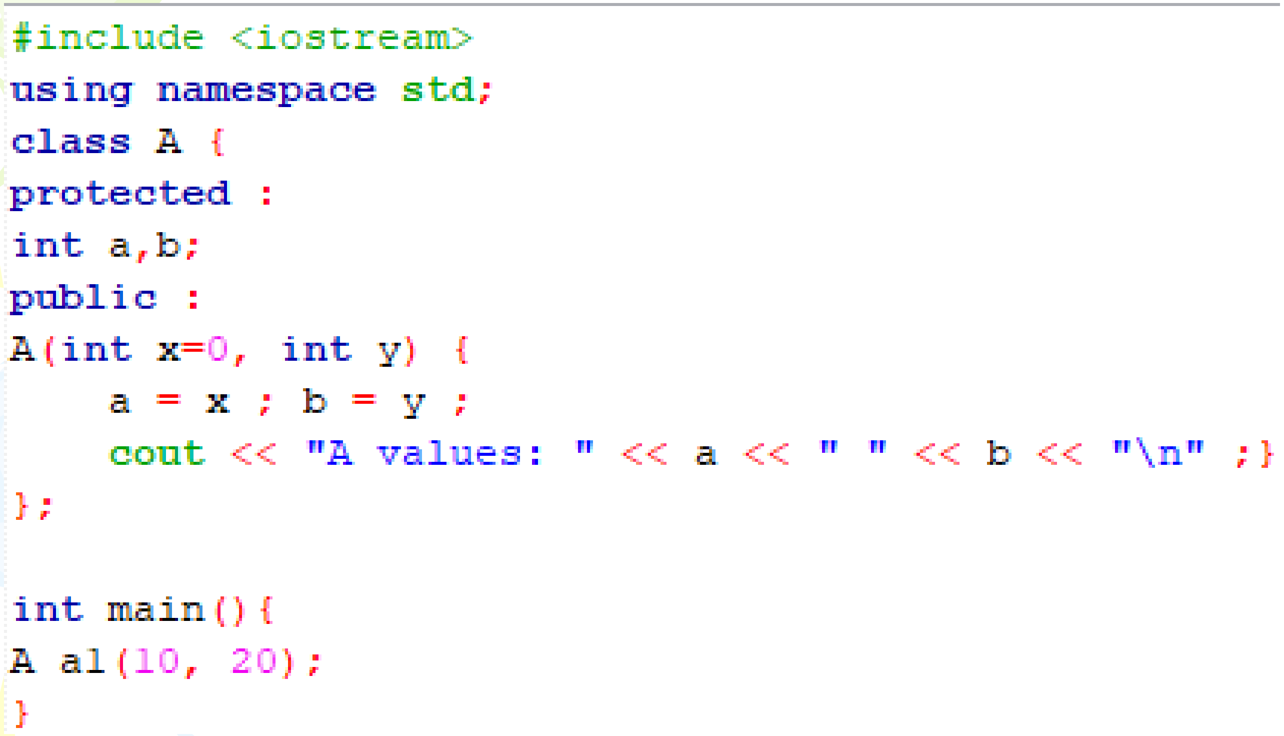
- #include <iostream>
- using namespace std;

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

What is the output of the following code?



```
#include <iostream>
using namespace std;
class A {
protected :
int a,b;
public :
A(int x=0, int y) {
    a = x ; b = y ;
    cout << "A values: " << a << " " << b << "\n" ;}
};

int main(){
A a1(10, 20);
}
```

[✓ A. Compilation Error]
B. Runtime Error
C. No error but nothing printed
D. A values: 10 20

Class Function Overloading

Refe
00:3

- A **default parameter** is a function parameter that has a default value provided to it. Example,

```
void PrintValues(int nValue1, int nValue2=10)
{
    cout << "1st value: " << nValue1 << endl;
    cout << "2nd value: " << nValue2 << endl;
}
 int main()
{
    PrintValues(1); // nValue2 will use default parameter of 10
    PrintValues(3, 4); // override default value for nValue2
}
```

Rules:

1) All default parameters must be the **rightmost** parameters. The following is not allowed:

```
void PrintValue(int nValue1=10, int nValue2); // not allowed
```

```
#include <iostream.h>

class A {
 protected :
       int a,b;
 public :
        A(int x=0, int y) {
               a = x ;
               b = y ;
        }
        virtual void print() ;


};
class B: public A {
 private:
         float p,q ;
 public :
        B(int m, int n, float u, float v) {
              p = u ;
              q = v ;
        }
        B() { p = q = 0 ; }
        void input(float u, float v) ;
        virtual void print(float) ;
};
```

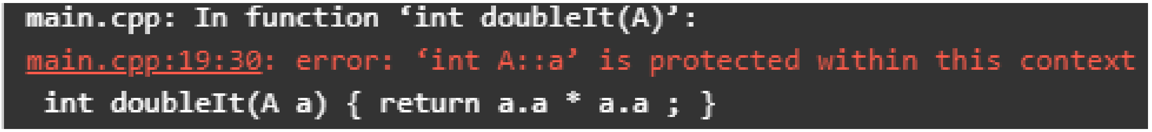
slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

What is the output of the following code?

```
#include <iostream>
using namespace std;
class A {
    protected :
    int a,b;
    public :
    A(int x, int y) {    a = x ;    b = y ;    }
};

int doubleIt(A a) { return a.a * a.a ; }

int main() {
  A aobj(2,3);
}
```



```
main.cpp: In function ‘int doubleIt(A)’:
main.cpp:19:30: error: ‘int A::a’ is protected within this context
 int doubleIt(A a) { return a.a * a.a ; }
```

✓ A. Compilation Error

B. Runtime Error

C. No error but nothing printed

Friend

Re
28

Friend allows non-member function access to private data of a class.

```
class Complex {
      double _real, _imag;  // private
     public:
            ....
     friend Complex operator +( const Complex , const Complex  );
};
 Complex operator +(const  Complex op1, const Complex op2) {
              double real = op1._real + op2._real ;
              double imag = op1._imag + op2._imag;
   return(Complex(real, imag));
}
```

- **Should not use friend unless necessary**
  - **Break the data hiding principle.**
  - **If used often it is a sign that it is time to restructure your inheritance.**

```
// friend class SomeClass;
```

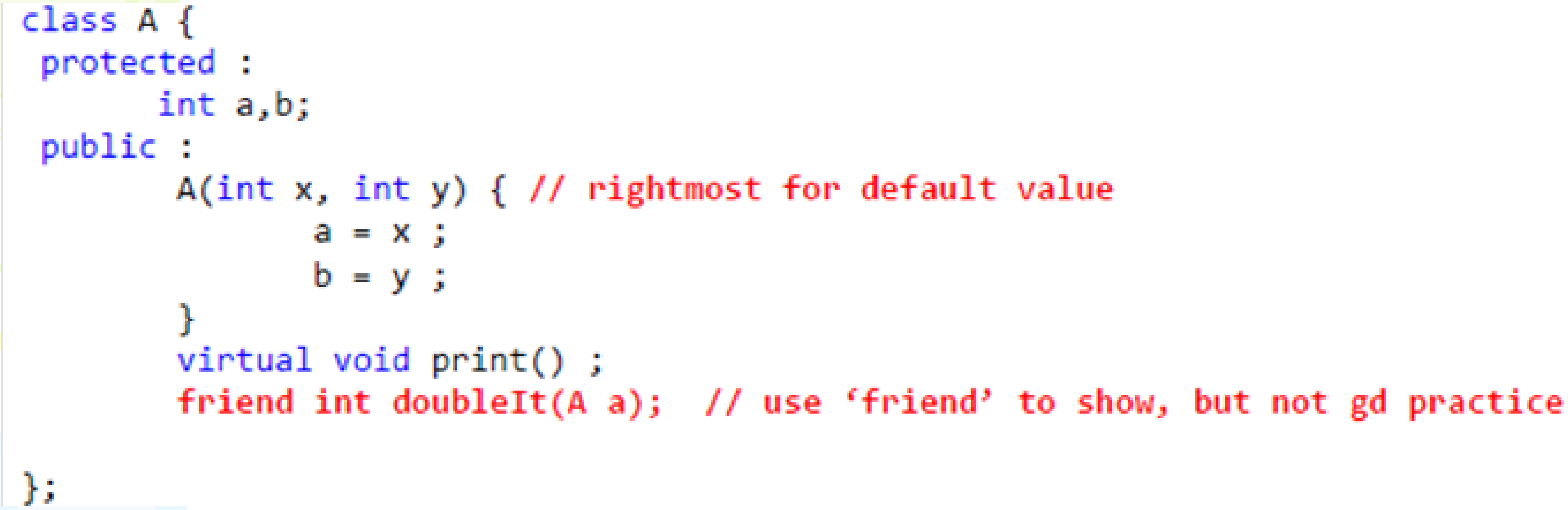
friend

Non-member functions of a class will not have access to the private/ protected data of another class. There could be situations where we want two classes to share some functions and the data members. In that case, we can make the function a friend of these classes, and that will enable the function to access the private and protected data members of the classes.

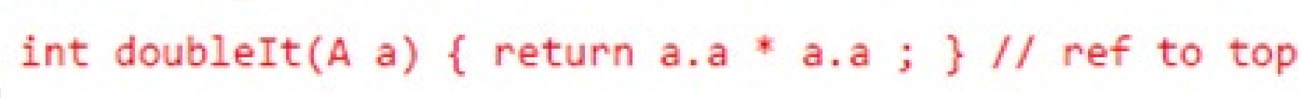
```
class className{
   //  Other Declarations
   friend returnType functionName(arg list);
};
```

```
#include <iostream.h>

class A {
 protected :
        int a,b;
 public :
         A(int x=0, int y) {
                a = x ;
                b = y ;
         }
         virtual void print() ;


};
class B: public A {
 private:
          float p,q ;
 public :
         B(int m, int n, float u, float v) {
               p = u ;
               q = v ;
         }
         B() { p = q = 0 ; }
         void input(float u, float v) ;
         virtual void print(float) ;
};
```

```
void A::print(void) {
       cout << "A values: " << a << " " << b << "\n" ;
}
void B::print(float) {
       cout << "B values : " << u << " " << v << "\n" ;
}
void B::input(float x, float y) {
       p = x ;   q = y ;
}

int doubleIt(A a) { return a.a * a.a ; }
```



```
class A {
 protected :
       int a,b;
 public :
        A(int x, int y) { // rightmost for default value
               a = x ;
               b = y ;
        }
        virtual void print() ;
        friend int doubleIt(A a);  // use ‘friend’ to show, but not gd practice

};
```



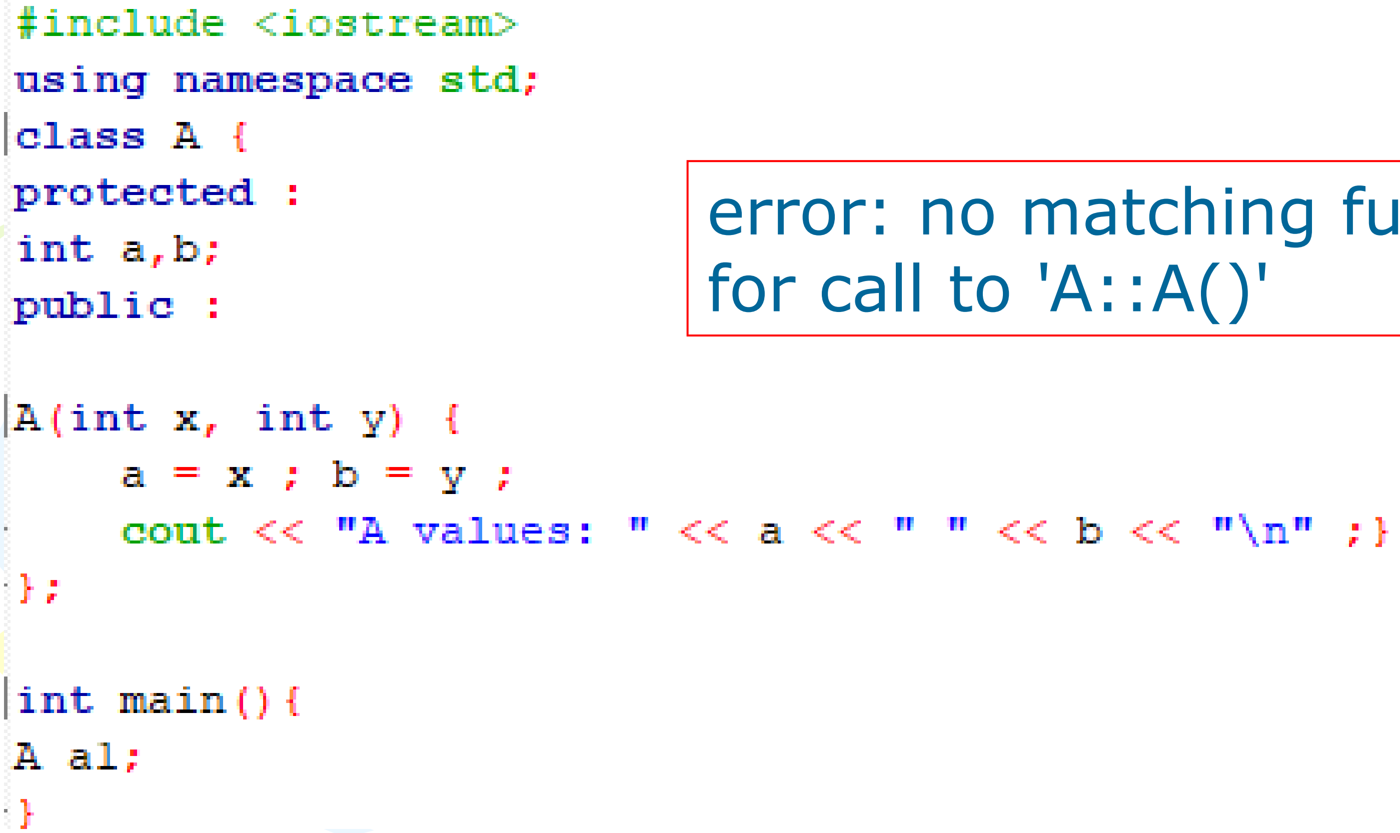
```
int doubleIt(A a) { return a.a * a.a ; } // ref to top
```

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

What is the output of the following code?



```
#include <iostream>
using namespace std;
class A {
protected :
int a,b;
public :

A(int x,  int y)  {
     a = x ; b = y ;
     cout << "A values: " << a << " " << b << "\n" ;}
};

int main(){
A a1;
}
```

error: no matching function for call to 'A::A()'

✓ A. Compilation Error

B. Runtime Error

C. No error but nothing printed

D. A values: 0 0

In C++, default constructor include the user defined constructor that with full default parameter, so can call by A() although there is some parameter defined. As long as user define a constructor, the system will not create the automatic A().

The system will generate automatic A() empty constructor only when there is ZERO constructor defined in class.

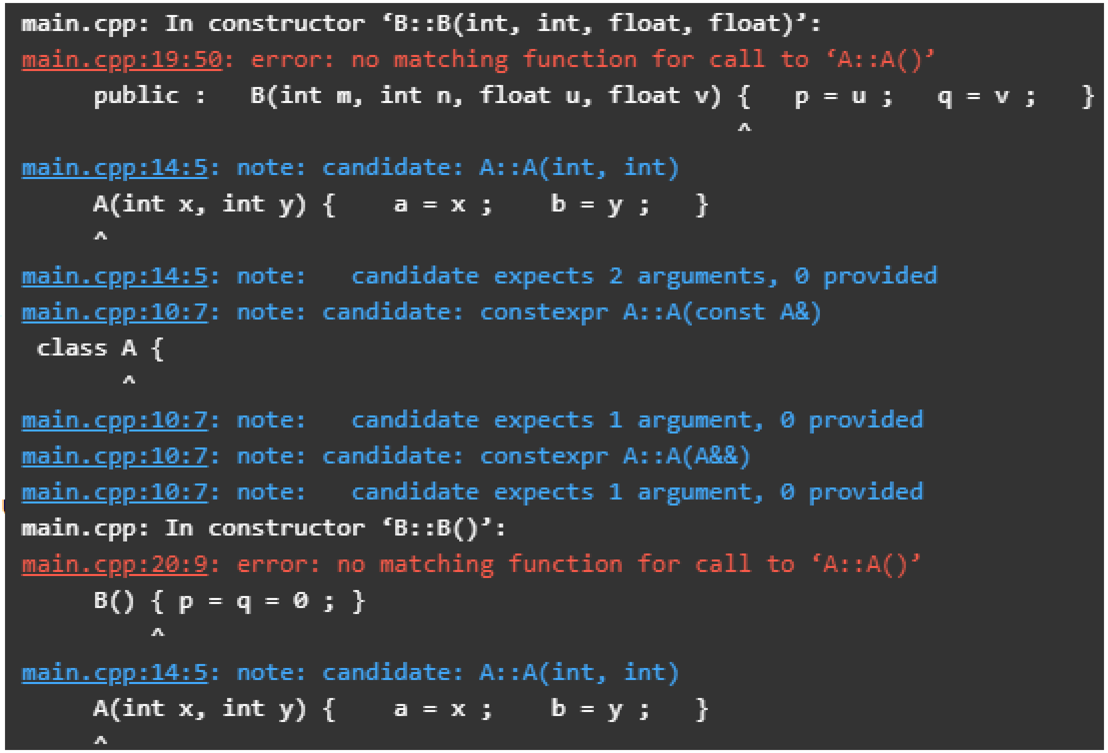
slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

What is the output of the following code?

```
#include <iostream>
using namespace std;
class A {
    protected :
    int a,b;
    public :
    A(int x, int y) {    a = x ;    b
};

class B: public A {
    private:    float p,q ;
    public :   B(int m, int n, float
    B() { p = q = 0 ; }
};

int main() {
}
```



```
main.cpp: In constructor ‘B::B(int, int, float, float)’:
main.cpp:19:50: error: no matching function for call to ‘A::A()’
     public :   B(int m, int n, float u, float v) {   p = u ;    q = v ;   }
                                                  ^
main.cpp:14:5: note: candidate: A::A(int, int)
     A(int x, int y) {    a = x ;    b = y ;   }
     ^
main.cpp:14:5: note:   candidate expects 2 arguments, 0 provided
main.cpp:10:7: note: candidate: constexpr A::A(const A&)
 class A {
       ^
main.cpp:10:7: note:   candidate expects 1 argument, 0 provided
main.cpp:10:7: note: candidate: constexpr A::A(A&&)
main.cpp:10:7: note:   candidate expects 1 argument, 0 provided
main.cpp: In constructor ‘B::B()’:
main.cpp:20:9: error: no matching function for call to ‘A::A()’
     B() { p = q = 0 ; }
         ^
main.cpp:14:5: note: candidate: A::A(int, int)
     A(int x, int y) {    a = x ;    b = y ;   }
     ^
```

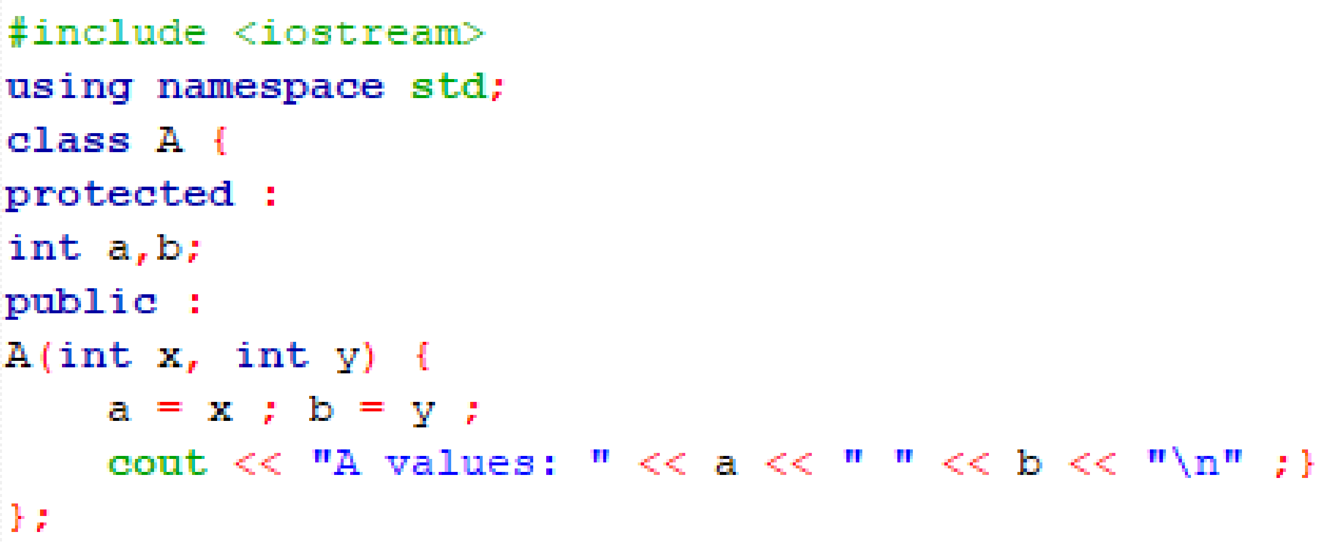
✓ A. Compilation Error

B. Runtime Error

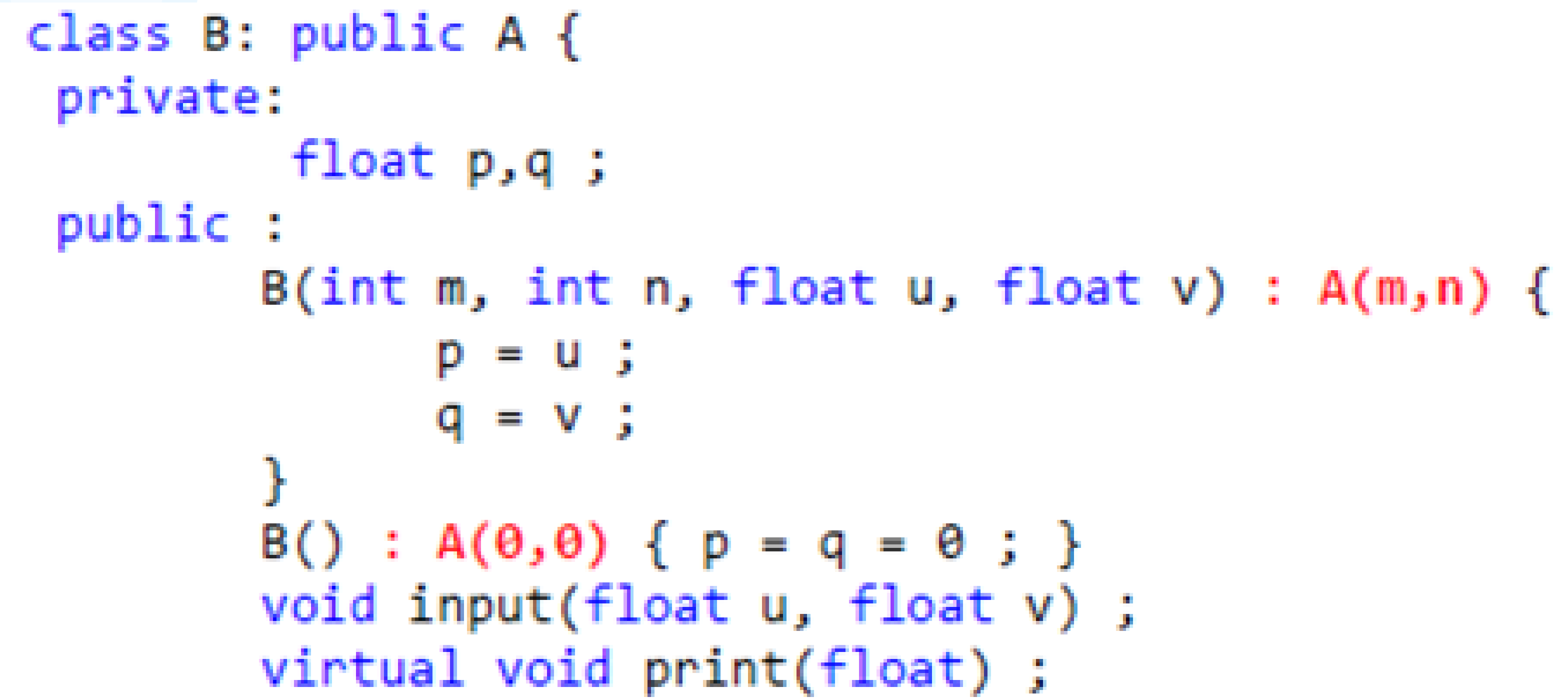
C. No error but nothing printed

```
#include <iostream.h>

class A {
 protected :
        int a,b;
 public :
         A(int x=0, int y) {
                a = x ;
                b = y ;
         }
         virtual void print() ;


};
class B: public A {
 private:
          float p,q ;
 public :
         B(int m, int n, float u, float v) {
               p = u ;
               q = v ;
         }
         B() { p = q = 0 ; }
         void input(float u, float v) ;
         virtual void print(float) ;
};
```

```
void A::print(void) {
       cout << "A values: " << a << " " << b << "\n" ;
}
void B::print(float) {
       cout << "B values : " << u << " " << v << "\n" ;
}
void B::input(float x, float y) {
       p = x ;   q = y ;
}

int doubleIt(A a) { return a.a * a.a ; }
```

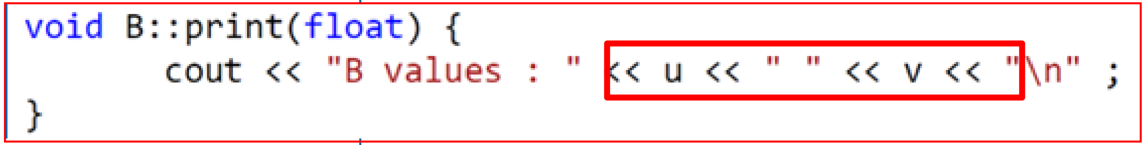


```
#include <iostream>
using namespace std;
class A {
protected :
int a,b;
public :
A(int x, int y) {
    a = x ; b = y ;
    cout << "A values: " << a << " " << b << "\n" ;}
};
```



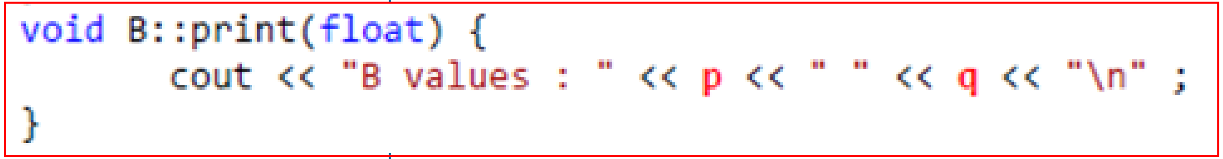
```
class B: public A {
 private:
         float p,q ;
 public :
        B(int m, int n, float u, float v) : A(m,n) {
              p = u ;
              q = v ;
        }
        B() : A(0,0) { p = q = 0 ; }
        void input(float u, float v) ;
        virtual void print(float) ;
```

```
#include <iostream.h>

class A {
 protected :
        int a,b;
 public :
         A(int x=0, int y) {
                a = x ;
                b = y ;
         }
         virtual void print() ;


};
class B: public A {
 private:
          float p,q ;
 public :
         B(int m, int n, float u, float v) {
               p = u ;
               q = v ;
         }
         B() { p = q = 0 ; }
         void input(float u, float v) ;
         virtual void print(float) ;
};
```



```
void B::print(float) {
       cout << "B values : " << u << " " << v << "\n" ;
}
```

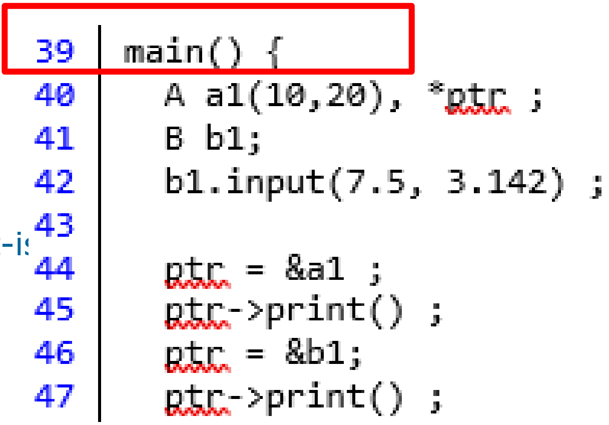
```
#include <iostream.h>

class A {
 protected :
        int a,b;
 public :
         A(int x=0, int y) {
                a = x ;
                b = y ;
         }
         virtual void print() ;


};
class B: public A {
 private:
          float p,q ;
 public :
         B(int m, int n, float u, float v) {
               p = u ;
               q = v ;
         }
         B() { p = q = 0 ; }
         void input(float u, float v) ;
         virtual void print(float) ;
};
```



```
void B::print(float) {
        cout << "B values : " << p << " " << q << "\n" ;
}
```

- **The return type of main must be int. No other return type is allowed**
- **default is int**
- It is very common to see incorrect programs that declare main with a return type of void; this is probably the most frequently violated rule concerning the main function.

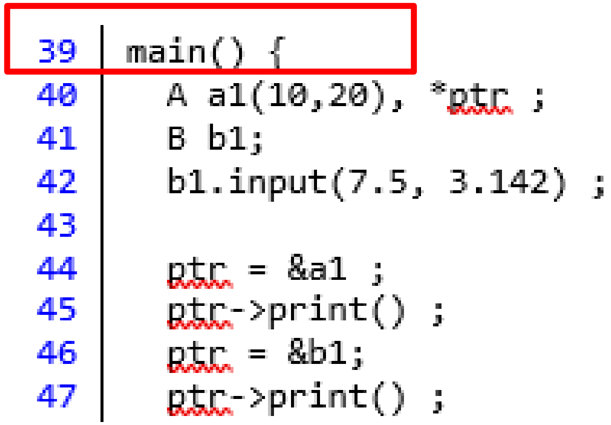
https://stackoverflow.com/questions/4207134/what-i



```
main() {
   A a1(10,20), *ptr ;
   B b1;
   b1.input(7.5, 3.142) ;

   ptr = &a1 ;
   ptr->print() ;
   ptr = &b1;
   ptr->print() ;
```

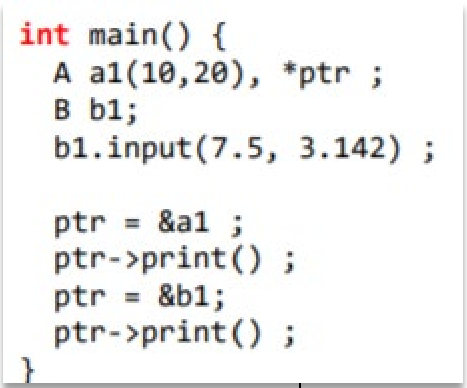
```
#include <iostream.h>

class A {
 protected :
       int a,b;
 public :
        A(int x=0, int y) {
                a = x ;
                b = y ;
        }
        virtual void print() ;


};
class B: public A {
 private:
         float p,q ;
 public :
        B(int m, int n, float u, float v) {
              p = u ;
              q = v ;
        }
        B() { p = q = 0 ; }
        void input(float u, float v) ;
        virtual void print(float) ;
};
```

```
void A::print(void) {
       cout << "A values: " << a << " " << b << "\n" ;
}
void B::print(float) {
       cout << "B values : " << u << " " << v << "\n" ;
}
void B::input(float x, float y) {
       p = x ;   q = y ;
}

int doubleIt(A a) { return a.a * a.a ; }
```



```
main() {
   A a1(10,20), *ptr ;
   B b1;
   b1.input(7.5, 3.142) ;

   ptr = &a1 ;
   ptr->print() ;
   ptr = &b1;
   ptr->print() ;
```

```
#include <iostream>
using namespace std;
class A {
 protected :
       int a,b;
 public :
        A(int x, int y) { // rightmost for default value
               a = x ;
               b = y ;
        }
        virtual void print() ;
        friend int doubleIt(A a);  // use ‘friend’ to show, but not gd practice

};
class B: public A {
 private:
         float p,q ;
 public :
        B(int m, int n, float u, float v) : A(m,n) {
              p = u ;
              q = v ;
        }
        B() : A(0,0) { p = q = 0 ; }
        void input(float u, float v) ;
        virtual void print(float) ;
};
void A::print(void) {
       cout << "A values: " << a << " " << b << "\n" ;
}
void B::print(float) {
       cout << "B values : " << p << " " << q << "\n" ;
}
void B::input(float x, float y) {
       p = x ;   q = y ;
}

int doubleIt(A a) { return a.a * a.a ; } // ref to top

int main() {
```

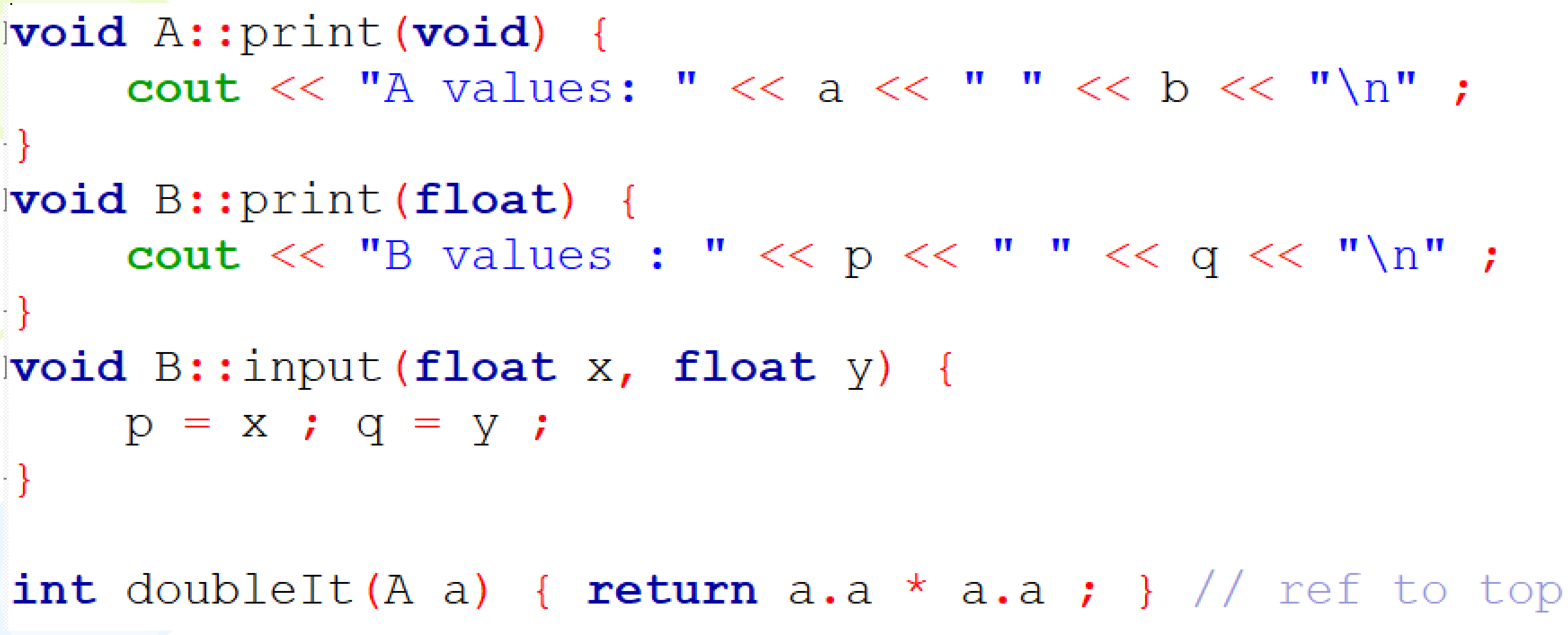


```
int main() {
  A a1(10,20), *ptr ;
  B b1;
  b1.input(7.5, 3.142) ;

  ptr = &a1 ;
  ptr->print() ;
  ptr = &b1;
  ptr->print() ;
}
```

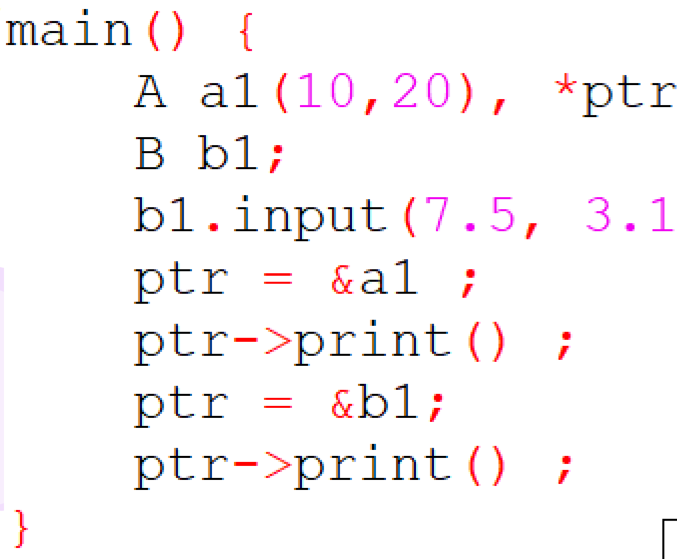
slido

What is the output of previous code?

ⓘ Start presenting to display the poll results on this slide.



```
void A::print(void) {
    cout << "A values: " << a << " " << b << "\n" ;
}
void B::print(float) {
    cout << "B values : " << p << " " << q << "\n" ;
}
void B::input(float x, float y) {
    p = x ; q = y ;
}

int doubleIt(A a) { return a.a * a.a ; } // ref to top
```



```
main() {
    A a1(10,20), *ptr ;
    B b1;
    b1.input(7.5, 3.142) ;
    ptr = &a1 ;
    ptr->print() ;
    ptr = &b1;
    ptr->print() ;
}
```

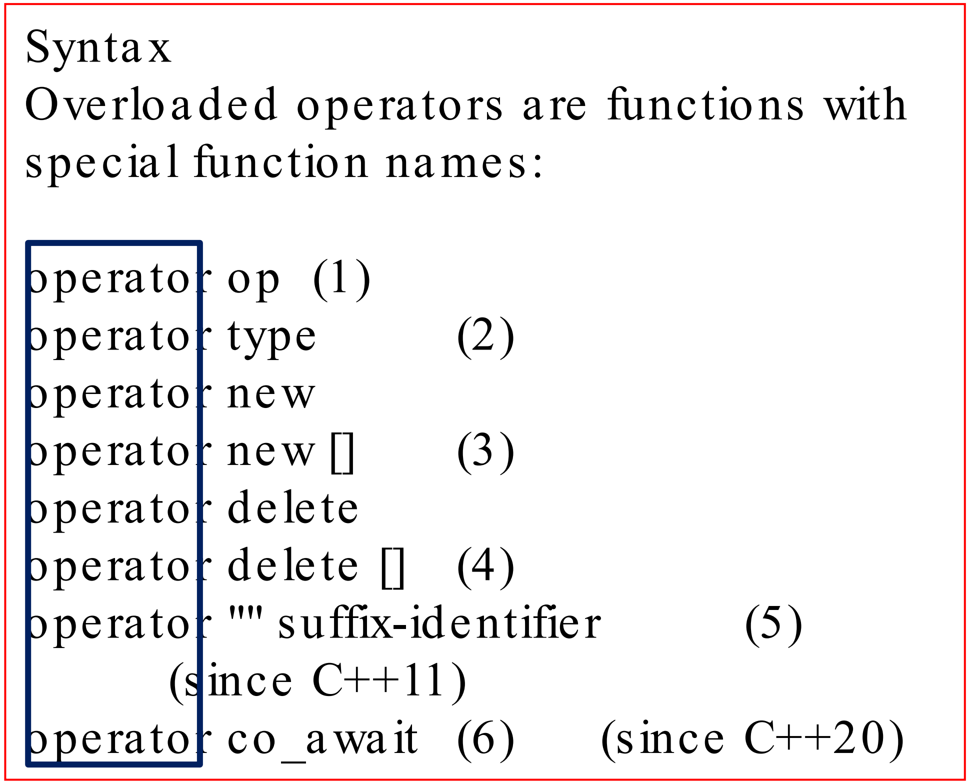
```
A values: 10 20
A values: 0 0
```

Overloading instead of overriding

1b

- (b) Write an operator overloaded method/function to add 2 objects of class A together and return the result as class A object.

Operator overloading is a crucial concept in C++ that lets you achieve the functionality of the built-in operators while working with user-defined data types. Comparison operators in C++ are the ones that are there to compare two values with each other such as “==”, “!=”, “>”, “<”, “>=”, and “<=”.



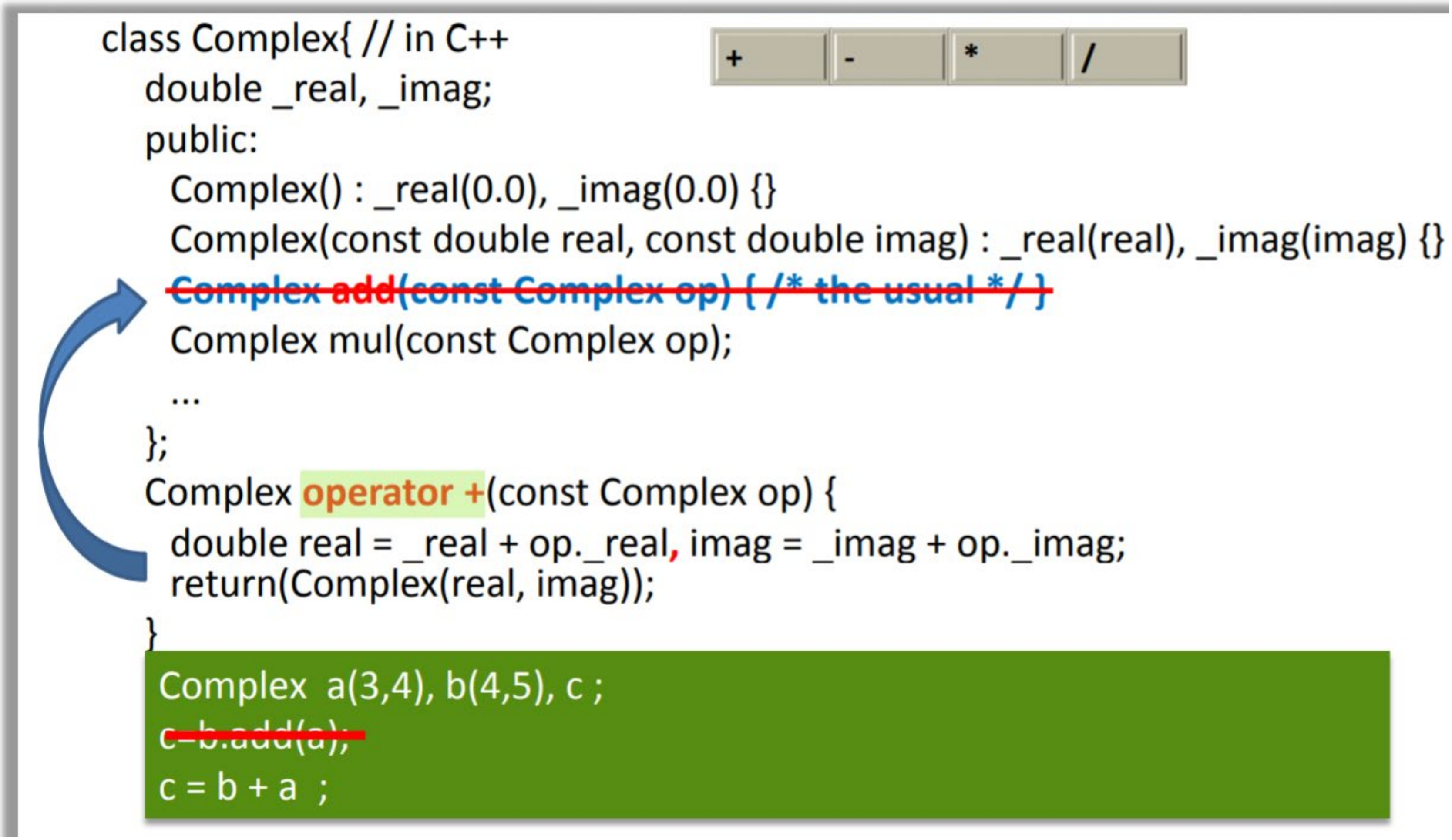
Syntax

Overloaded operators are functions with special function names:

```
operator op  (1)
operator type         (2)
operator new
operator new []       (3)
operator delete
operator delete []    (4)
operator "" suffix-identifier          (5)
        (since C++11)
operator co_await   (6)      (since C++20)
```

https:/ / linuxhint.com/ cpp-overload-comparison-operator-in/

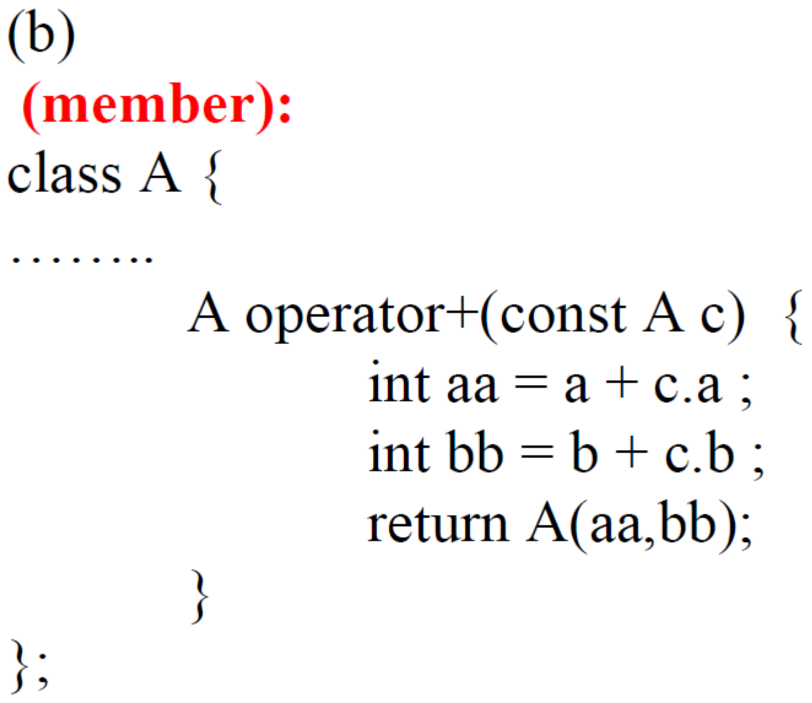
Operator Overloading



```
class Complex{ // in C++
    double _real, _imag;
    public:
      Complex() : _real(0.0), _imag(0.0) {}
      Complex(const double real, const double imag) : _real(real), _imag(imag) {}
      ~~Complex add(const Complex op) { /* the usual */ }~~
      Complex mul(const Complex op);
      ...
    };
    Complex operator +(const Complex op) {
      double real = _real + op._real, imag = _imag + op._imag;
      return(Complex(real, imag));
    }
```

```
Complex  a(3,4), b(4,5), c ;
~~c=b.add(a);~~
c = b + a  ;
```

(b) Write an operator overloaded method/function to add 2 objects of class A together and return the result as class A object.



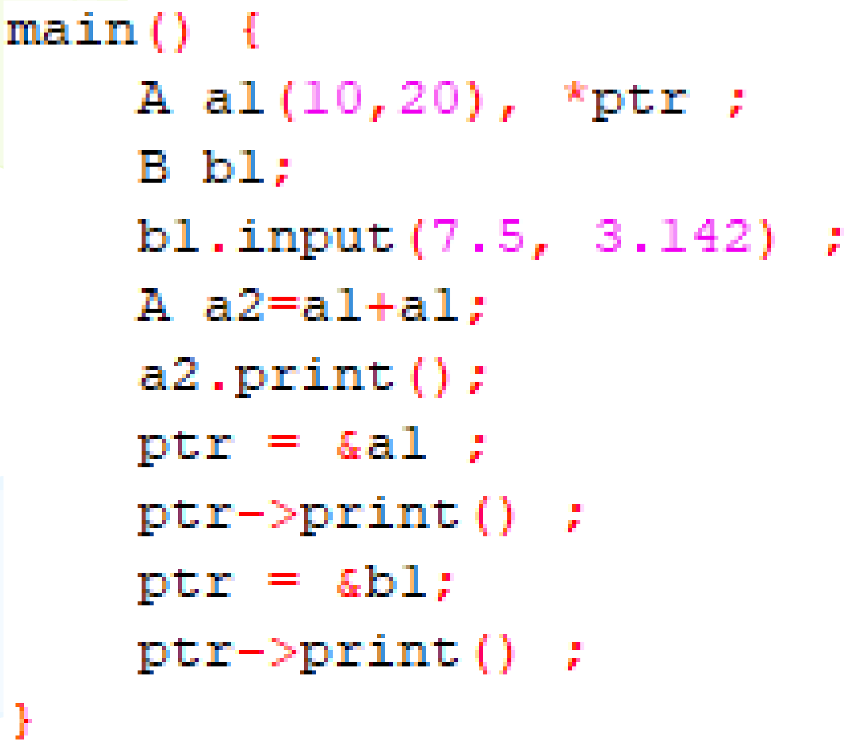
(b)

(member):

```
class A {
……..
        A operator+(const A c)  {
                int aa = a + c.a ;
                int bb = b + c.b ;
                return A(aa,bb);
        }
};
```

<u>Or</u> (non-member)

```
class A {
………
        friend A operator+(const A z, const A y) ;
};
         A operator+(const A z, const A y)  {
               int aa = y.a + z.a ;
               int bb = y.b + z.b ;
               return A(aa,bb);
         }
```



```
main()  {
        A a1(10,20), *ptr ;
        B b1;
        b1.input(7.5, 3.142) ;
        A a2=a1+a1;
        a2.print();
        ptr = &a1 ;
        ptr->print() ;
        ptr = &b1;
        ptr->print() ;
}
```

```
A values: 20 40
A values: 10 20
A values: 0 0
```

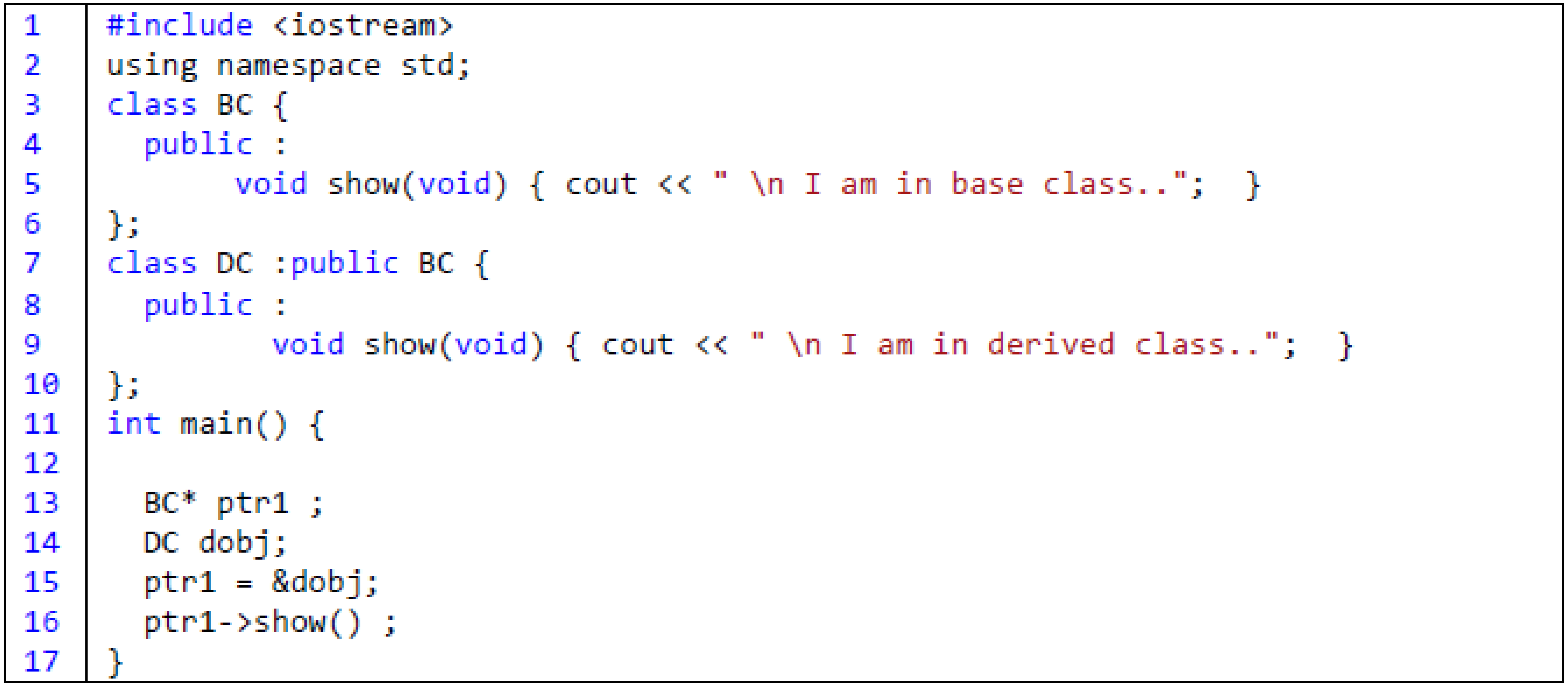
Q2

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

What is the output of the following code?



```
#include <iostream>
using namespace std;
class BC {
  public :
       void show(void) { cout << " \n I am in base class..";  }
};
class DC :public BC {
  public :
         void show(void) { cout << " \n I am in derived class..";  }
};
int main() {

  BC* ptr1 ;
  DC dobj;
  ptr1 = &dobj;
  ptr1->show() ;
}
```

A. Compilation Error

B. Runtime Error

C. I am in derived class..

✓ D. I am in base class..

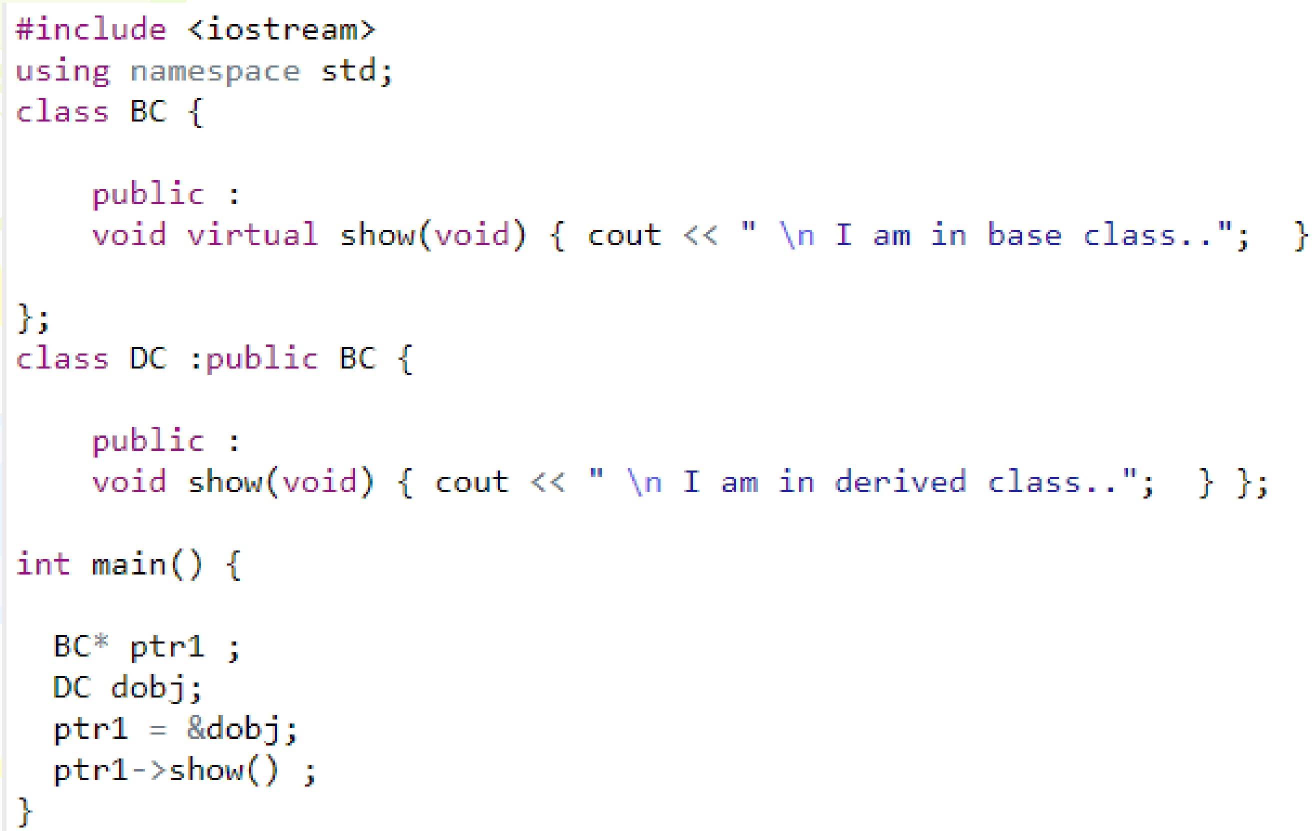
E. No error but nothing printed

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

What is the output of the following code?



```
#include <iostream>
using namespace std;
class BC {

    public :
    void virtual show(void) { cout << " \n I am in base class..";  }

};
class DC :public BC {

    public :
    void show(void) { cout << " \n I am in derived class..";  } };

int main() {

  BC* ptr1 ;
  DC dobj;
  ptr1 = &dobj;
  ptr1->show() ;
}
```

A. Compilation Error

B. Runtime Error

[✓ C. I am in derived class..]

D. I am in base class..

E. No error but nothing printed

Polymorphism

- **Virtual**
  - To force method evaluation to be based on object type rather than reference type. [<ref type> <name> = new <obj type>(..)]
  - Without **virtual =>** non polymorphic (no dynamic binding)
  - **Example :** **virtual** void area() { cout << "........" << endl ; }
  - **Virtual** function magic only operates on pointers(*) and references(&).
  - If a method is declared **virtual** in a class, it is **automatically** **virtual** in all **derived** classes.
- **Pure method =>** **abstract** method (pure virtual)
  - By placing **"= 0"** in its declaration
  - **Example :** **virtual** void area() **= 0** ; // abstract method
  - **The class becomes an** **abstract** class