


- Solution

This lesson explains the solution to the exercise in the previous lesson.

WE'LL COVER THE FOLLOWING ^

- Solution
- Explanation

Solution

 Explicit

 Implicit

```
#include <iostream>

class A{};

class B{};

class MyClass{
public:
    MyClass(){}
    explicit MyClass(A){}           // since C++98
    explicit operator B(){return B();} // new with C++11
};

void needMyClass(MyClass){};
void needB(B){};

struct MyBool{
    explicit operator bool(){return true;}
};

int main(){

    // A -> MyClass
    A a;

    // explicit invocation
    MyClass myClass1(a);
    // implicit conversion from A to MyClass
    MyClass myClass2=a;
    needMyClass(a);

    // MyClass -> B
    MyClass myCl
```

```

MyClass myCl;

// explicit invocation

B b1(myCl);
// implicit conversion from MyClass to B
B b2= myCl;
needB(myCl);

// MyBool -> bool conversion
MyBool myBool;
if (myBool){};
int myNumber = (myBool)? 1998: 2011;
// implicit conversion
int myNewNumber = myBool + myNumber;
auto myTen = (20*myBool -10*myBool)/myBool;

std::cout << myTen << std::endl;
}

```



Explanation

- Recall that the **explicit** keyword is solely responsible for preventing implicit conversions.
- Hence, the trick is to simply remove **explicit** from the conversion constructor and operator. This will enable implicit conversions again.

That brings us to the end of this topic. Next on our list is the **call operator**.