CSCI 2312 OOP

Constructors
& about `const` & `static` modifiers

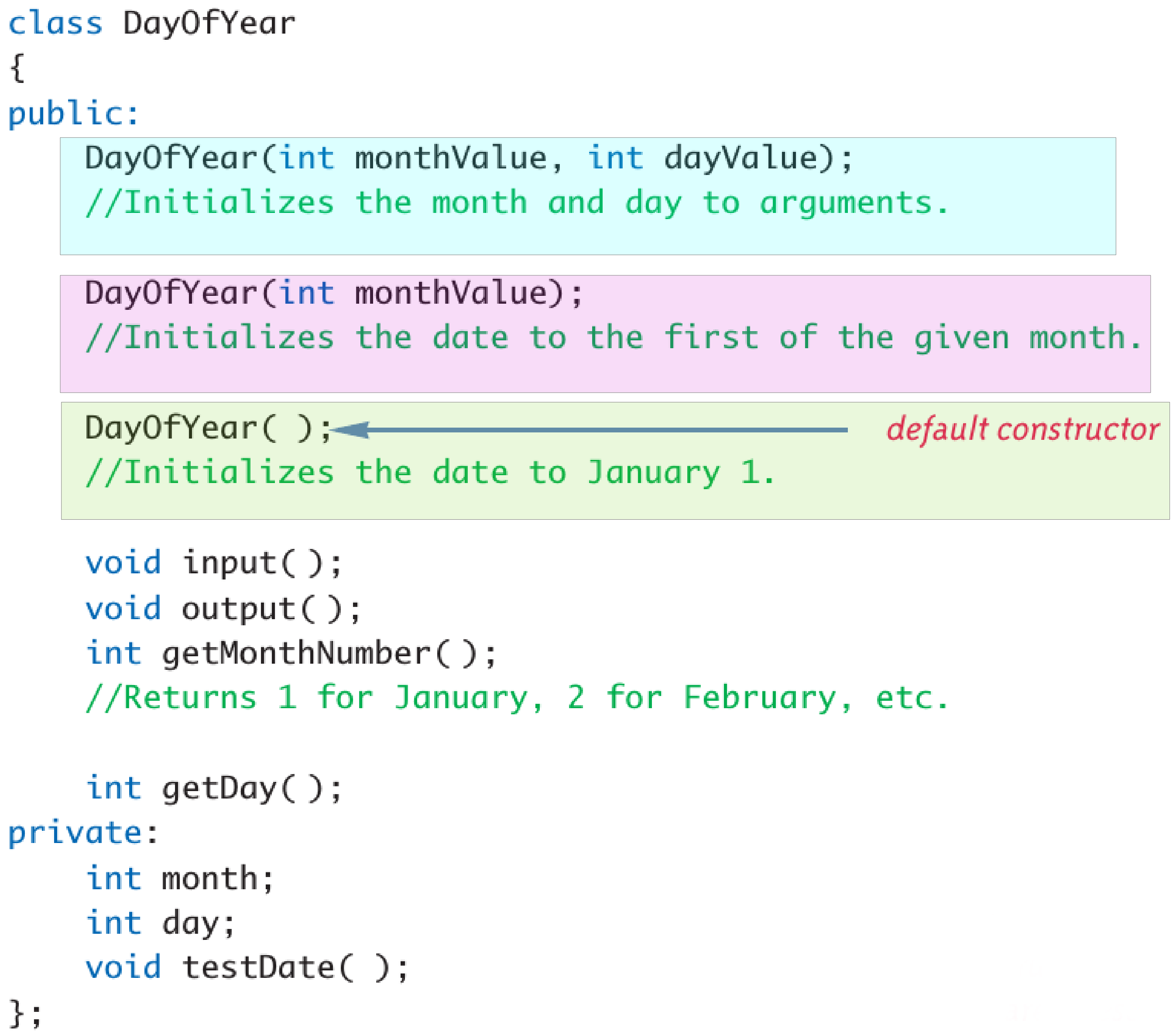
Textbook reference: Chapter 7

What is a constructor

- It is a member function of a class that initializes the instance of the class (i.e., the class object)
- A constructor has:
  - The same name as the class itself
  - no return type, *not even void.*
- ***Demo:***
  - *4A.cpp [default constructor = a constructor that takes no argument]*
    - *If there is no constructor in a class, compiler creates one default constructor for you.*
  - *4B.cpp [0, 1, and 2-argument constructors]*
    - ***Question: what happens now to the compiler generated default constructor?***
      - *Since your class already defined at least one constructor, now compiler does nothing for you.*

n-argument constructors (n=0,1,...)

- Demo:
  - 4B.cpp



```
class DayOfYear
{
public:
    DayOfYear(int monthValue, int dayValue);
    //Initializes the month and day to arguments.

    DayOfYear(int monthValue);
    //Initializes the date to the first of the given month.

    DayOfYear( );                          default constructor
    //Initializes the date to January 1.

    void input( );
    void output( );
    int getMonthNumber( );
    //Returns 1 for January, 2 for February, etc.

    int getDay( );
private:
    int month;
    int day;
    void testDate( );
};
```

```cpp
DayOfYear::DayOfYear(int monthValue, int dayValue)
{
    month = monthValue;
    day = dayValue;
}
```

OR,

```cpp
DayOfYear::DayOfYear(int monthValue, int dayValue)
                               : month(monthValue), day(dayValue)
{/*Body intentionally empty*/}
```

```
DayOfYear::DayOfYear(int monthValue) : month(monthValue), day(1)
{

}

DayOfYear::DayOfYear( ) : month(1), day(1)
{/*Body intentionally empty.*/}
```

When & how to call the constructors

- When: during you want to declare a class object.
- How?
  - Explicitly call the corresponding constructors during declaration

```
DayOfYear date1(2, 21), date2(5), date3;
```

Default constructor

- A constructor that takes no arguments is called a default constructor.
- It is important to remember not to use parentheses when you declare a class variable and want the default constructor be called.

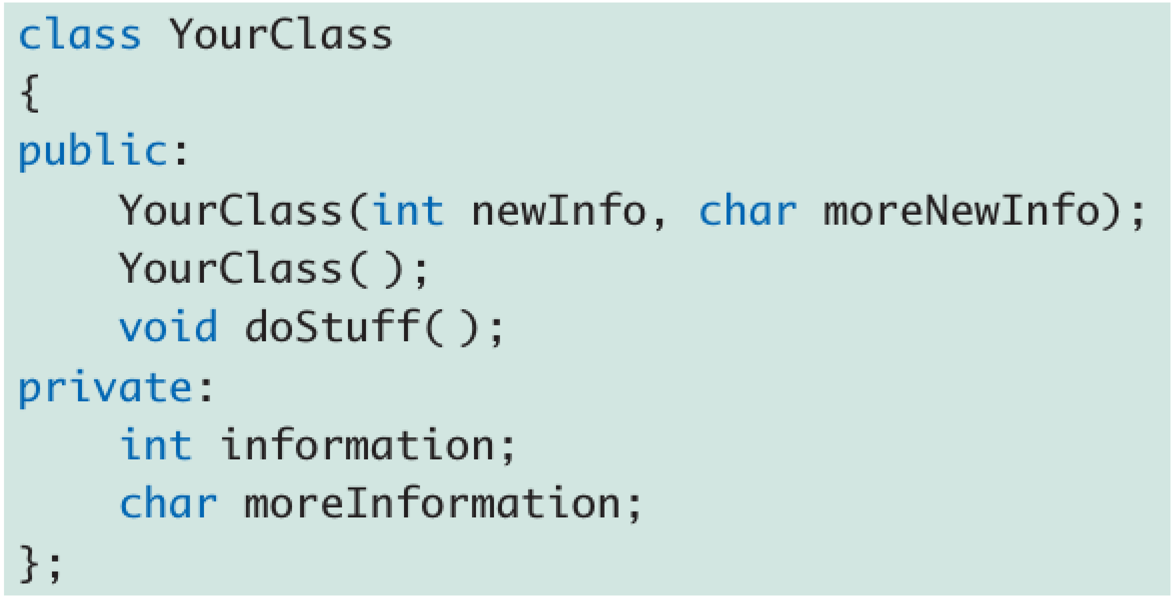
```
DayOfYear date2;          // OK
```

```
DayOfYear date2( );//PROBLEM!
```

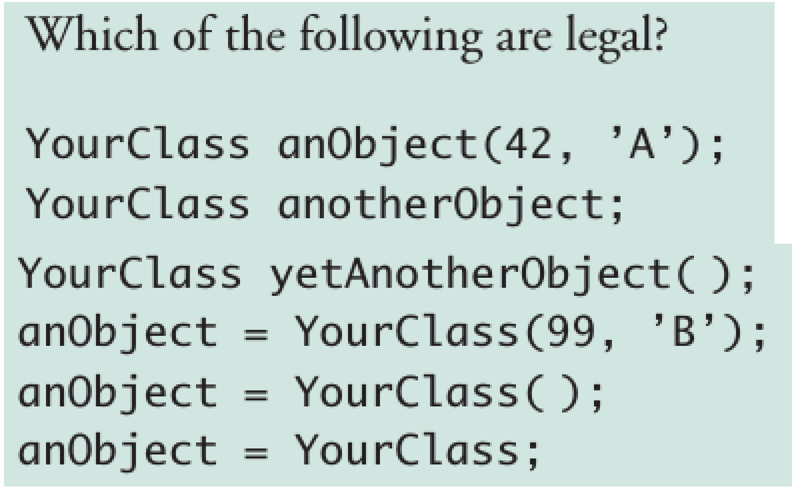
(The problem is that this syntax declares a function that returns a DayOfYear object and has no parameters.)

```
date1 = DayOfYear( );     // OK
```

Exercise 1



```
class YourClass
{
public:
    YourClass(int newInfo, char moreNewInfo);
    YourClass( );
    void doStuff( );
private:
    int information;
    char moreInformation;
};
```

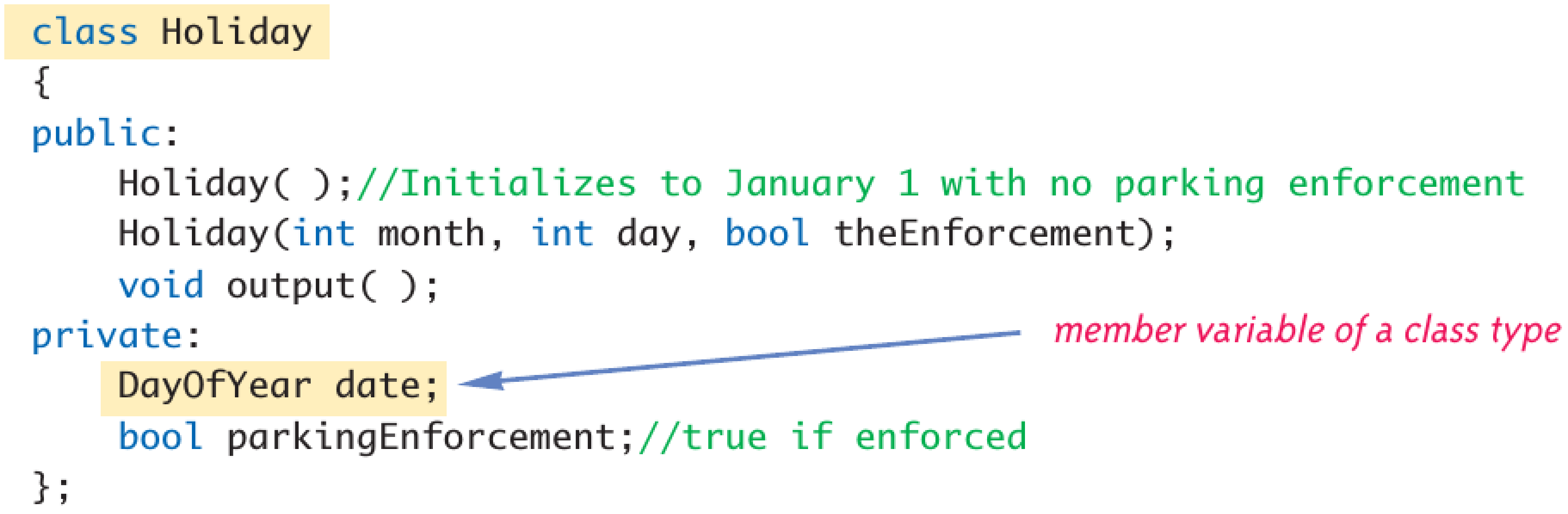


Which of the following are legal?

```
YourClass anObject(42, 'A');
YourClass anotherObject;
YourClass yetAnotherObject( );
anObject = YourClass(99, 'B');
anObject = YourClass( );
anObject = YourClass;
```

Class type member variables

- Also known as **composition** ===== introducing "**Has-a**" relationship between classes
- A class may have a member variable whose type is that of another class.
  - There is nothing special that you need to do to have a class member variable, but you must invoke the constructor of the member class from within the host class's constructor.



```
class Holiday
{
public:
    Holiday( );//Initializes to January 1 with no parking enforcement
    Holiday(int month, int day, bool theEnforcement);
    void output( );
private:
    DayOfYear date;
    bool parkingEnforcement;//true if enforced
};
```

member variable of a class type

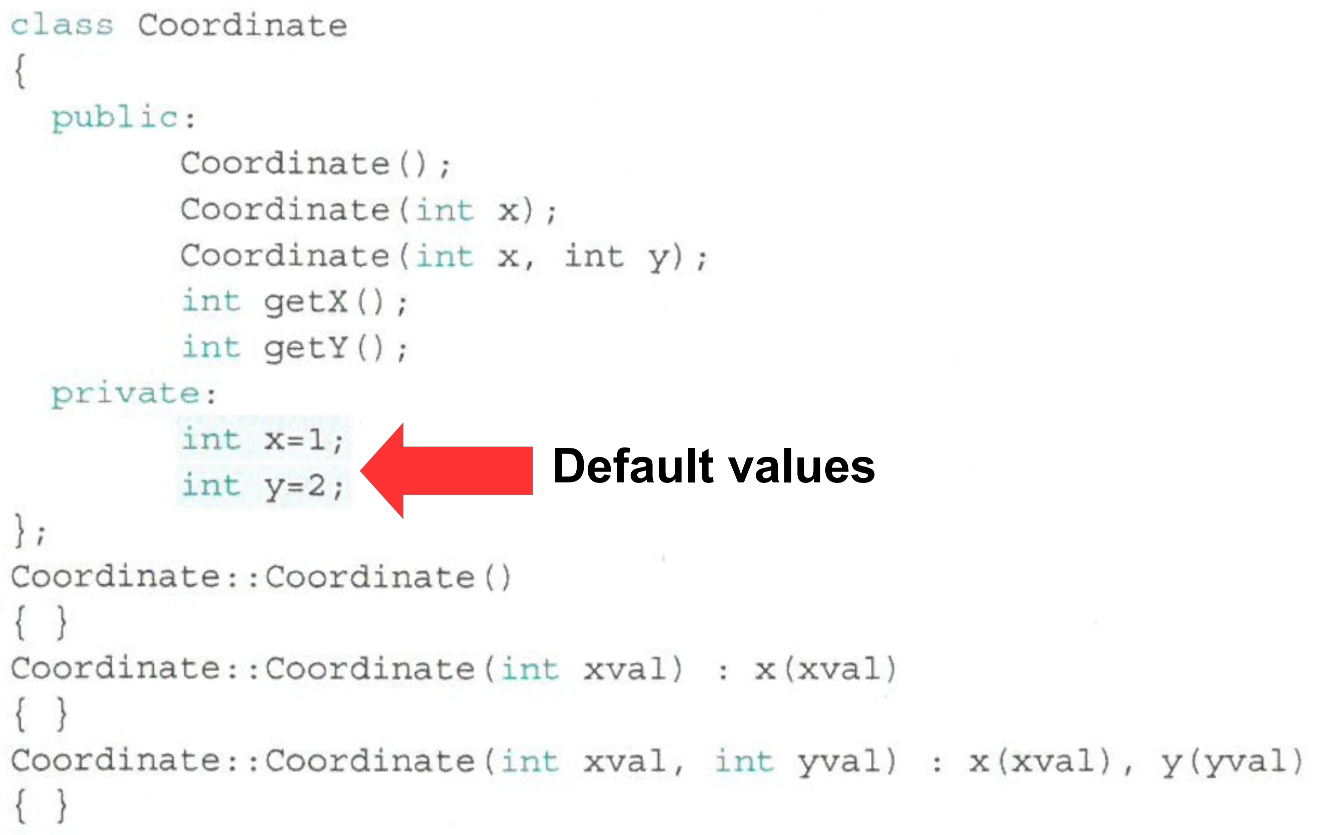
Invocations of constructor of a member class

Invocations of constructors

```
Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
{/*Intentionally empty*/}

Holiday::Holiday(int month, int day, bool theEnforcement)
                    : date(month, day), parkingEnforcement(theEnforcement)
{/*Intentionally empty*/}
```

Default values to class members

- This feature allows you to set default values for the member variables.



```
class Coordinate
{
  public:
          Coordinate();
          Coordinate(int x);
          Coordinate(int x, int y);
          int getX();
          int getY();
  private:
          int x=1;
          int y=2;
};
Coordinate::Coordinate()
{ }
Coordinate::Coordinate(int xval) : x(xval)
{ }
Coordinate::Coordinate(int xval, int yval) : x(xval), y(yval)
{ }
```

Default values

Constructor delegation

- This allows one constructor to call another constructor.

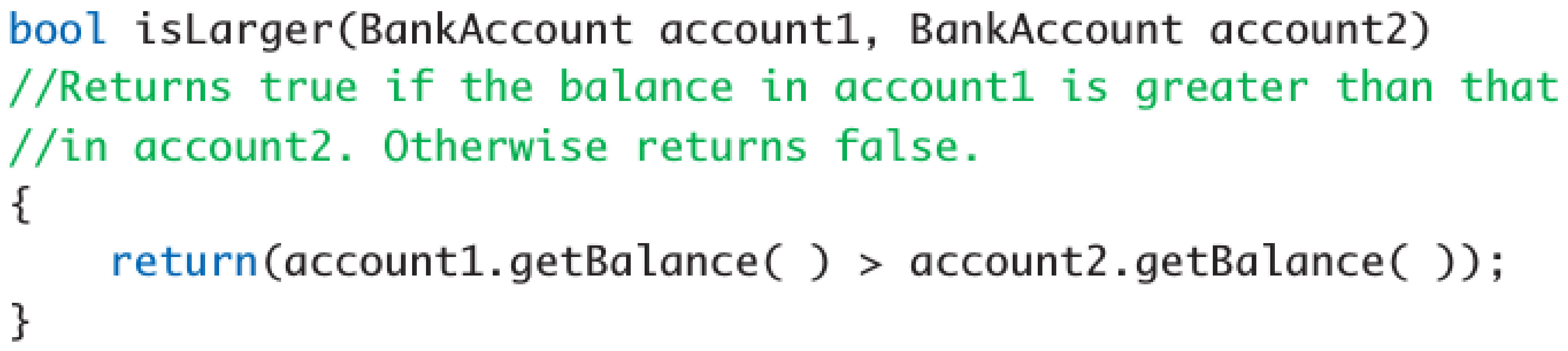
```
Coordinate::Coordinate() : Coordinate(99,99)
{


}
```

Now, let’s create an object of class Coordinate.

```
Coordinate c1;
//this will invoke the default constructor,
//  which in turn will invoke the corresponding
//  2-argument constructor to set x=99,y=99
```

`const` parameters

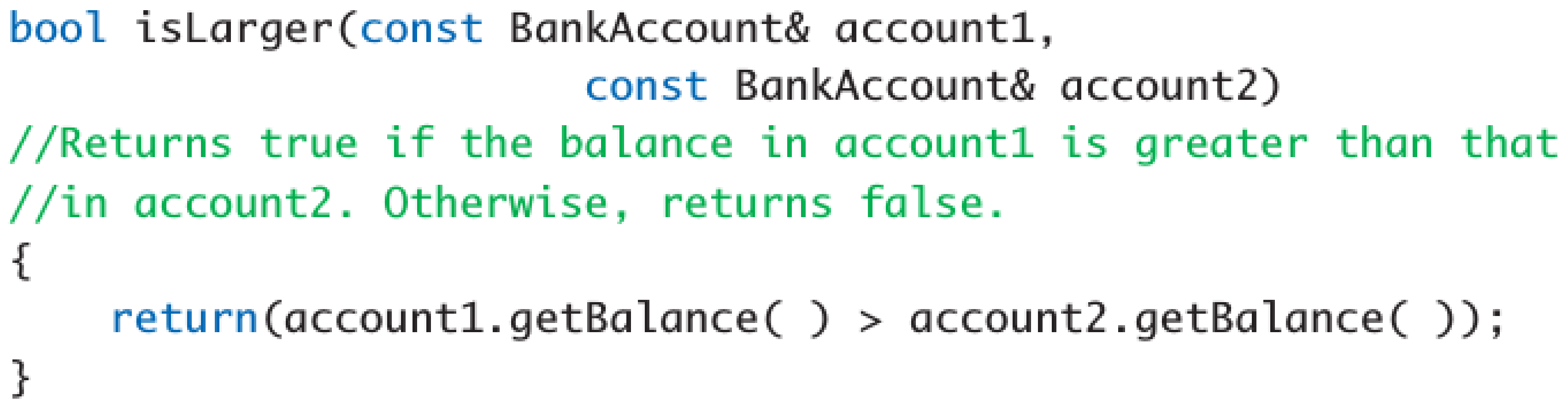
- If you are using call-by-reference parameter and your function does not change the values of the parameter, it is then safe to mark the parameter const to let the compiler know that the parameter won’t be changed in that function.



```
bool isLarger(BankAccount account1, BankAccount account2)
//Returns true if the balance in account1 is greater than that
//in account2. Otherwise returns false.
{
    return(account1.getBalance( ) > account2.getBalance( ));
}
```

`const` parameters

- If you are using call-by-reference parameter and your function does not change the values of the parameter, it is then safe to mark the parameter const to let the compiler know that the parameter won’t be changed in that function.



```
bool isLarger(const BankAccount& account1,
                          const BankAccount& account2)
//Returns true if the balance in account1 is greater than that
//in account2. Otherwise, returns false.
{
    return(account1.getBalance( ) > account2.getBalance( ));
}
```

const with member functions

- If you have a member function that should not change the value of a calling object, you can mark the function with `const` modifier.

```
class BankAccount
{
public:
        ...
    void output( ) const;
        ...
```

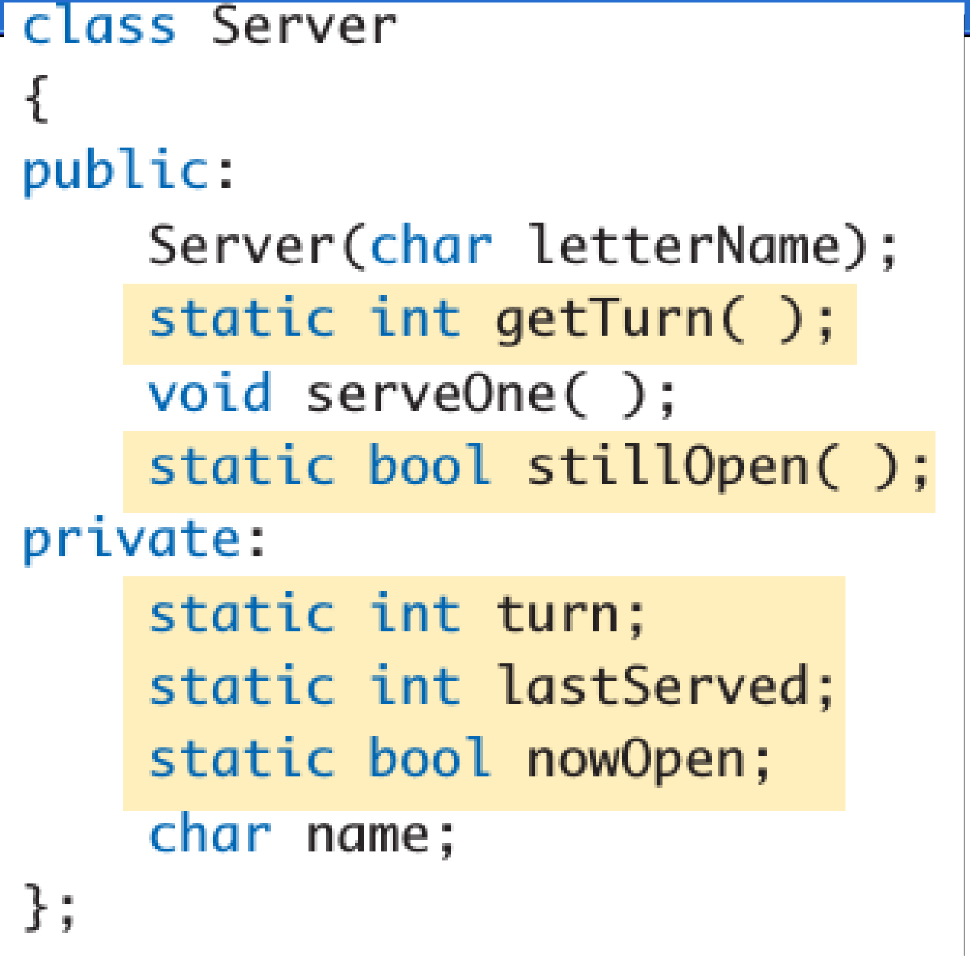
```
void BankAccount::output( ) const
{
    ...
```

any function call from this **output** function must also be **const**, otherwise the compiler will generate an error.

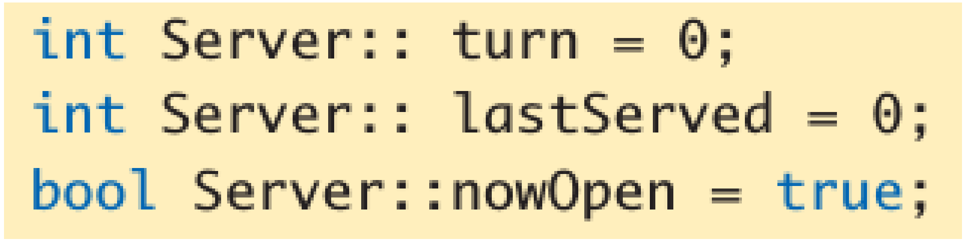
Static members

- Sometimes you want to have one variable that is shared by all the objects of a class.
  - Such variables are called **static variables.**
  - it allows some of the advantages of "global variables" without opening the flood gates to all the abuses that true global variables invite.
- If a function does not access the data of any object and yet you want the function to be a member of the class, you can make it a **static function**.
  - however, a static function has access to static variables of the class.

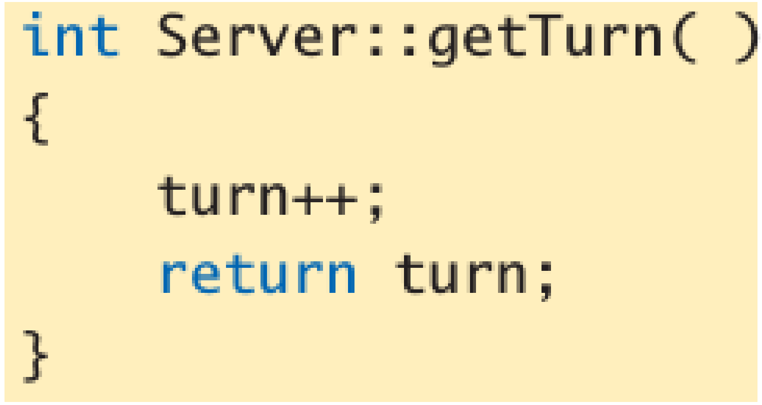
Static members



```
class Server
{
public:
    Server(char letterName);
    static int getTurn( );
    void serveOne( );
    static bool stillOpen( );
private:
    static int turn;
    static int lastServed;
    static bool nowOpen;
    char name;
};
```



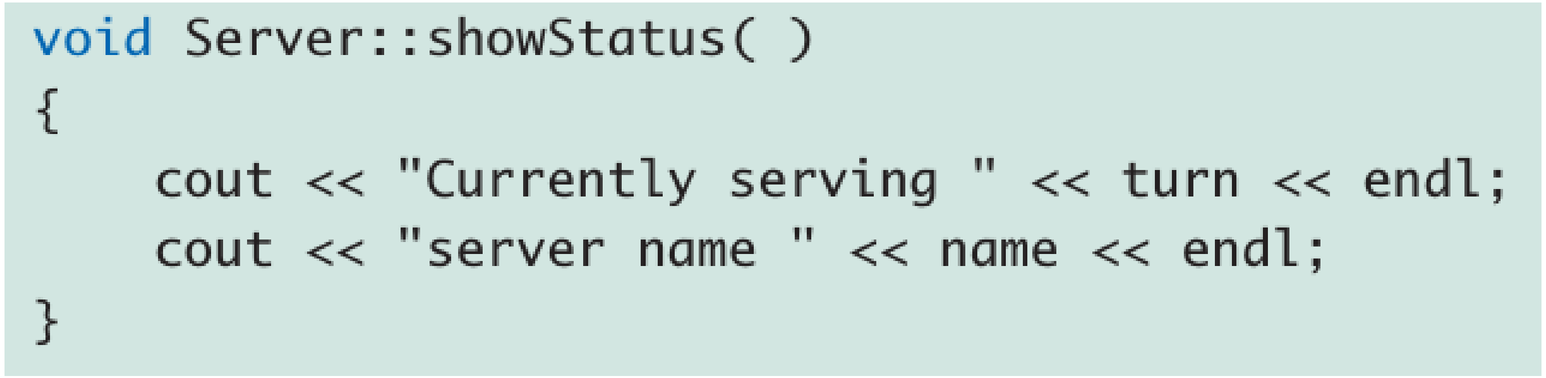
```
int Server:: turn = 0;
int Server:: lastServed = 0;
bool Server::nowOpen = true;
```



```
int Server::getTurn( )
{
    turn++;
    return turn;
}
```

Exercise

- Could the function defined as follows be added to the class Server as a static function?



```
void Server::showStatus( )
{
    cout << "Currently serving " << turn << endl;
    cout << "server name " << name << endl;
}
```