Deeper Look into Classes

LAB 6: Employee Count

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

3/24/2022

Objectives

- Learn how to use friend. (Chapter 10.4)
- Learn how to use static data members and static member functions. (Chapter 10.6)

Friend Functions & Friend Classes

- A friend function of a class is defined outside that class's scope, yet has the right to access the non-public members of the class. (Chapter 10.4)
- A friend can be a function, an entire class, or a member function of another class.
- Friendship relation is neither symmetric nor transitive.

```
class Count {
  friend void setX( Count &, int ); // friend
declaration
public:
  Count(): x( 0 ) // initialize x to 0
  { /* empty body*/ } // end constructor
private:
  int x;
};
void Count::print const { cout << x << endl; }
void setX(Count &c, int val)
{ c.x = val;}  // setX is a friend function
```

```
int main()
{
  Count counter; // create Count object
  cout << "counter.x after instantiation: ";
  counter.print();
  setX( counter, 8 );
 // set x using a friend function
  cout << "counter.x after call to setX friend
function: ";
  counter.print();
} // end main
```

Static Data Member

- **(Chapter 10.6)**
  - Used to handle the case that only one copy of the member's value for all the objects belonging to the same class. For example, a static data member called objectCount can be used to record the number of objects of the same type.
  - Belonging to no object. Thus, It should be initialized in the global namespace scope by **className::staticDataMember= initialValue.**
  - Being accessed using className::staticDataMember if it is a public one.
  - Being accessed using a public static member function or friends if it is a private or protected one.

Example: Static Data Member

- // Used when all objects refer to the same value of the data member.

```
#include <string>
using namespace std;
class Employee
{
public:
   Employee( const string &, const string & ); // constructor
   ~Employee(); // destructor
   string getFirstName() const; // return first name
   string getLastName() const; // return last name
   // static member function
   static int getCount(); // return numbers of objects instantiated
private:
   string firstName;
   string lastName;
   // static data
   static int count; // number of objects instantiated
}; // end class Employee
```

Initializing Static Data Members

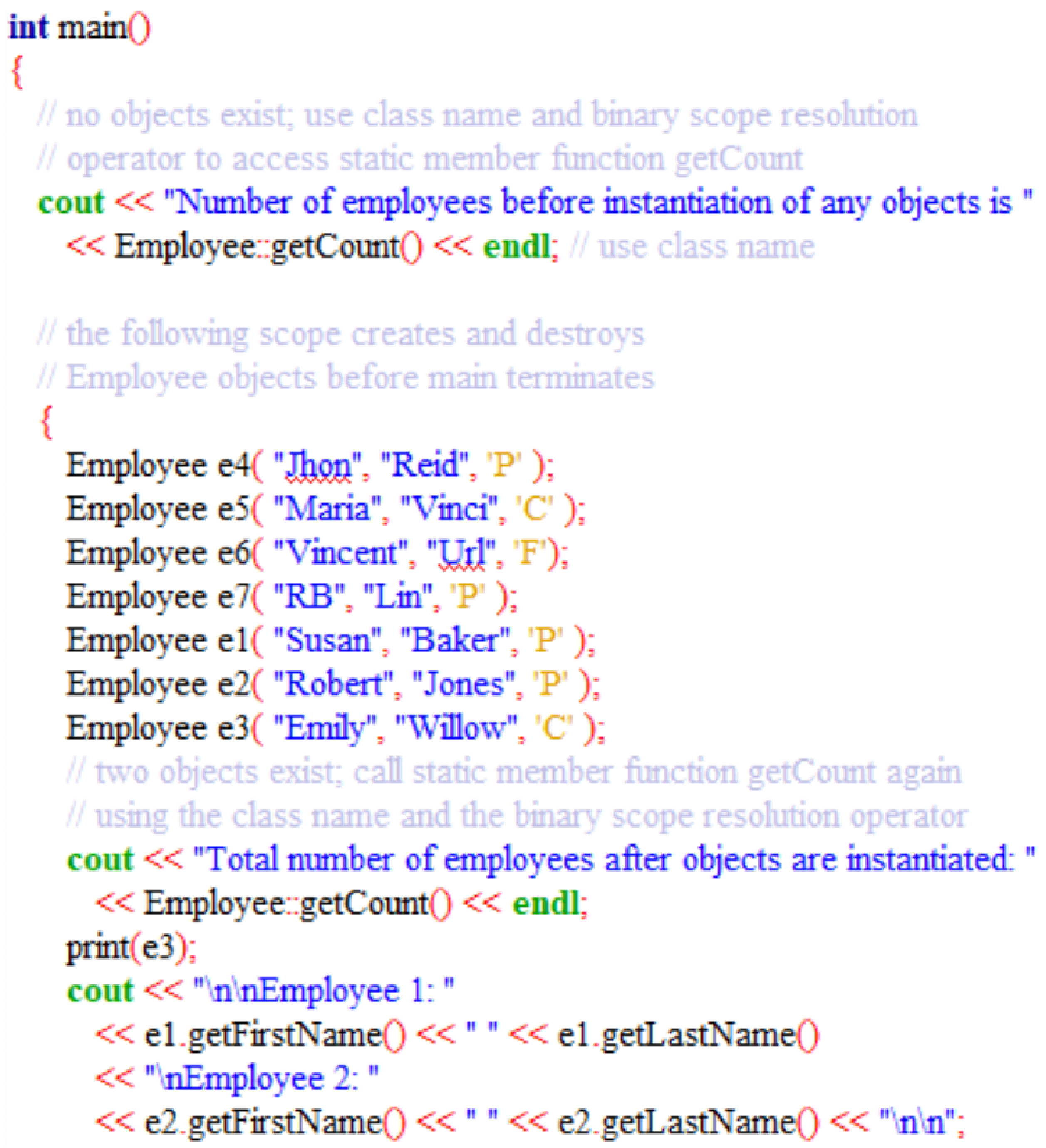
- Usually initialized in the global scope, i.e., not in the class definition.

```
// Fig. 10.21: Employee.cpp
// Employee class member-function definitions.
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

// define and initialize static data member at global namespace scope
int Employee::count = 0; // cannot include keyword static

// define static member function that returns number of
// Employee objects instantiated (declared static in Employee.h)
int Employee::getCount()
{
   return count;
} // end static function getCount
```

Lab 6: Employee Count

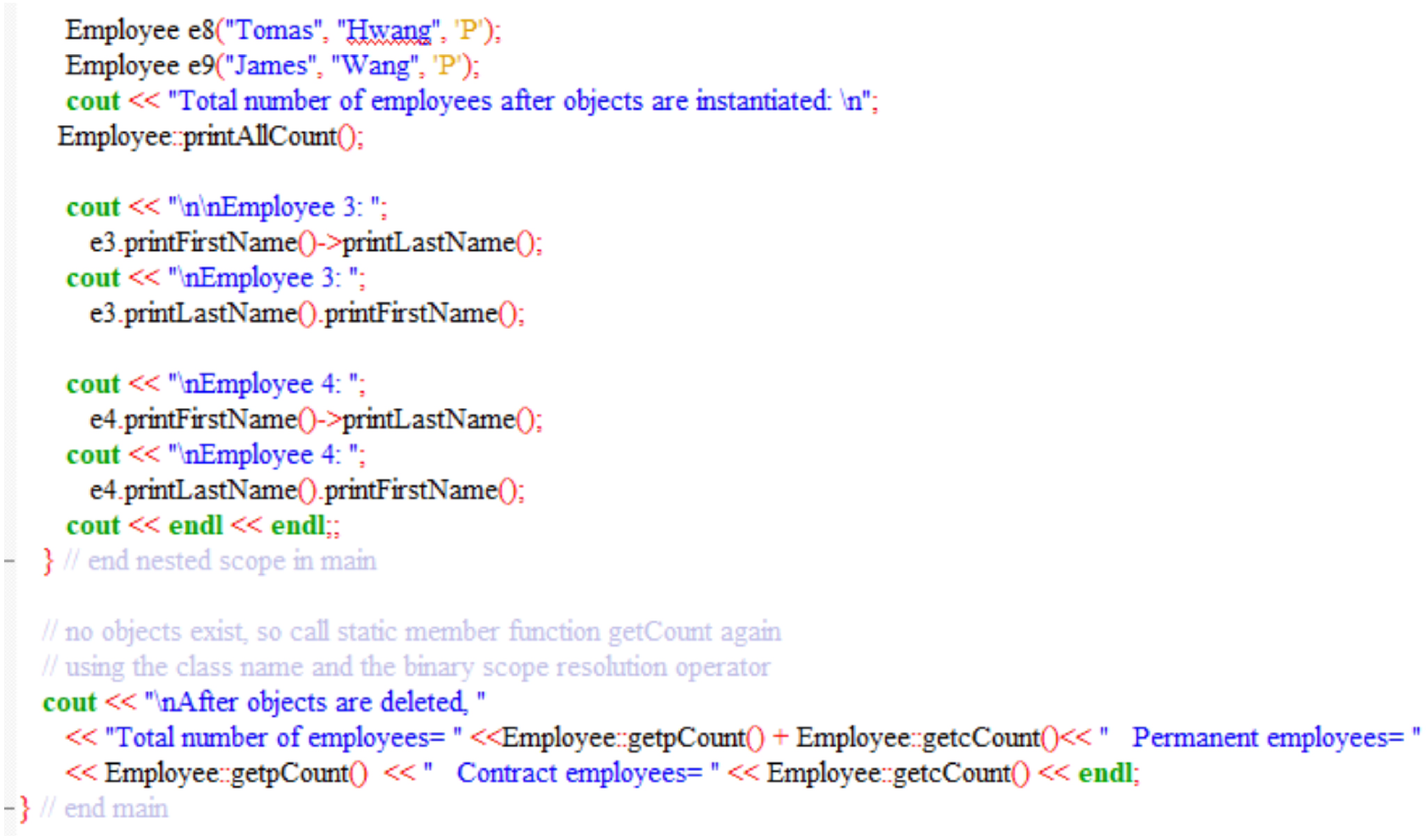
- **Modify the code in Fig. 10.20, Fig. 10.21 such that the given main() function will generate the desired output. The following are the major tasks needed to be completed.**
  - **Add a copy constructor Employee(const Employee &).**
  - **Add a private data member eType of type char. If eType is ‘P’, the employee is a permanent employee. If eType is ‘C’, the employee is a contract employee.**
  - **Replaced data member count with static data members pCount and cCount which counts the number of permanent employees and the number of contract employees, respectively.**
  - **Add a member function int getcCount() to return the value of cCount.**
  - **Add a member function int getpCount() to return the value of pCount.**
  - **Add a member function void printAllCount() to print the values of all the static data members of class Employee.**
  - **Add two member functions Employee* printFirstName() and Employee & printLastName() to print the first name and last name of an employee, respectively.**
  - **Add a function void print(const Employee &) to print the values of all the non-static data members of an employee in the global namespace scope and make it as a friend of the class Employee.**
- **The given main() function should not be changed.**

main() Function (1)



```
int main()
{
  // no objects exist; use class name and binary scope resolution
  // operator to access static member function getCount
  cout << "Number of employees before instantiation of any objects is "
    << Employee::getCount() << endl; // use class name

  // the following scope creates and destroys
  // Employee objects before main terminates
  {
    Employee e4( "Jhon", "Reid", 'P' );
    Employee e5( "Maria", "Vinci", 'C' );
    Employee e6( "Vincent", "Url", 'F');
    Employee e7( "RB", "Lin", 'P' );
    Employee e1( "Susan", "Baker", 'P' );
    Employee e2( "Robert", "Jones", 'P' );
    Employee e3( "Emily", "Willow", 'C' );
    // two objects exist; call static member function getCount again
    // using the class name and the binary scope resolution operator
    cout << "Total number of employees after objects are instantiated: "
      << Employee::getCount() << endl;
    print(e3);
    cout << "\n\nEmployee 1: "
      << e1.getFirstName() << " " << e1.getLastName()
      << "\nEmployee 2: "
      << e2.getFirstName() << " " << e2.getLastName() << "\n\n";
```

main() Function (2)

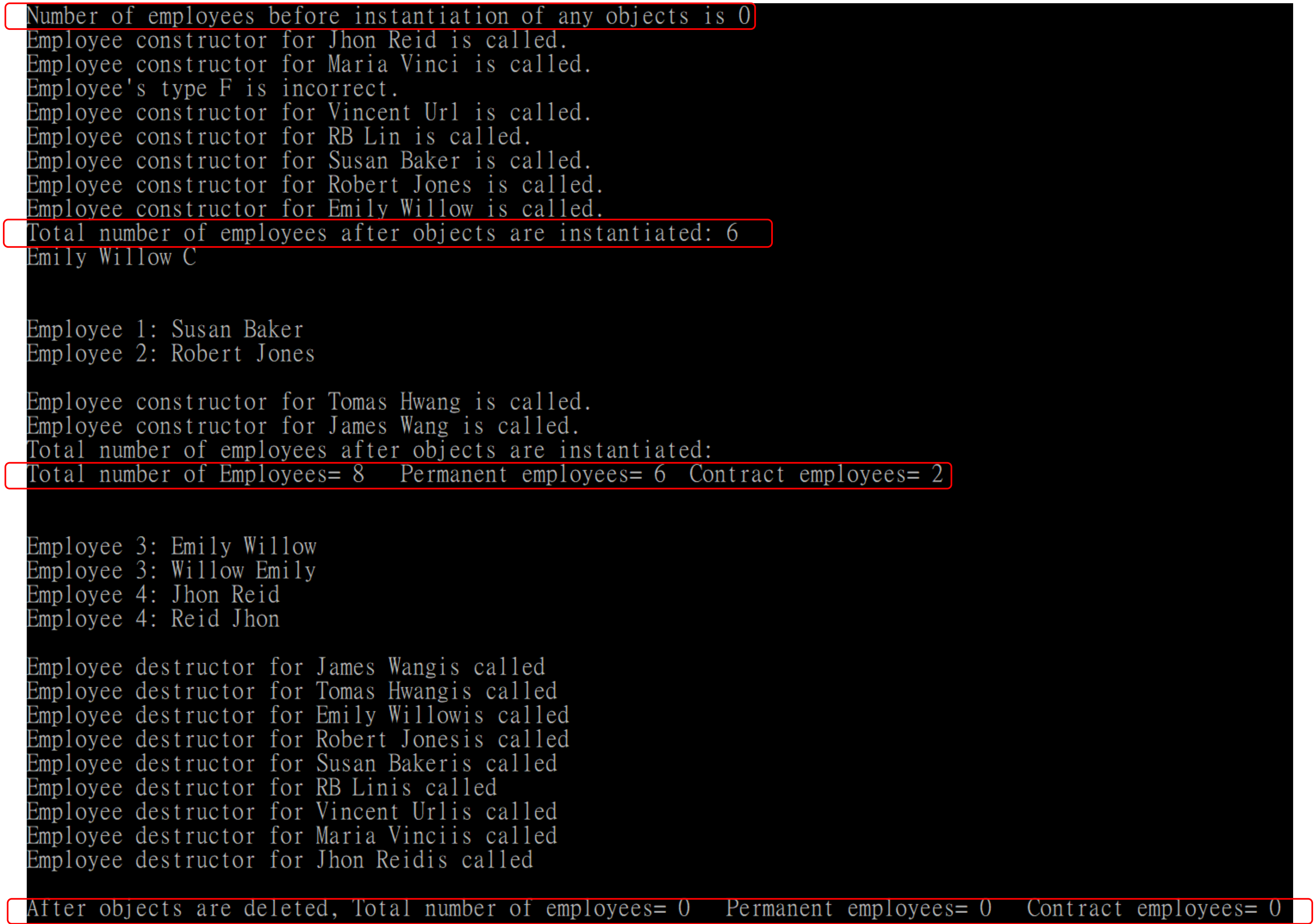


```
        Employee e8("Tomas", "Hwang", 'P');
        Employee e9("James", "Wang", 'P');
        cout << "Total number of employees after objects are instantiated: \n";
       Employee::printAllCount();

        cout << "\n\nEmployee 3: ";
          e3.printFirstName()->printLastName();
        cout << "\nEmployee 3: ";
          e3.printLastName().printFirstName();

        cout << "\nEmployee 4: ";
          e4.printFirstName()->printLastName();
        cout << "\nEmployee 4: ";
          e4.printLastName().printFirstName();
        cout << endl << endl;;
-     } // end nested scope in main

      // no objects exist, so call static member function getCount again
      // using the class name and the binary scope resolution operator
      cout << "\nAfter objects are deleted, "
        << "Total number of employees= " <<Employee::getpCount() + Employee::getcCount()<< "   Permanent employees= "
        << Employee::getpCount()  << "   Contract employees= " << Employee::getcCount() << endl;
-} // end main
```

Key Points for Grading

- Output should be the same as that given in this lab. Especially check the printout highlighted by red circles marked in the output.
- All the member functions and data members should be exactly implemented.

Output



```
Number of employees before instantiation of any objects is 0
Employee constructor for Jhon Reid is called.
Employee constructor for Maria Vinci is called.
Employee's type F is incorrect.
Employee constructor for Vincent Url is called.
Employee constructor for RB Lin is called.
Employee constructor for Susan Baker is called.
Employee constructor for Robert Jones is called.
Employee constructor for Emily Willow is called.
Total number of employees after objects are instantiated: 6
Emily Willow C


Employee 1: Susan Baker
Employee 2: Robert Jones

Employee constructor for Tomas Hwang is called.
Employee constructor for James Wang is called.
Total number of employees after objects are instantiated:
Total number of Employees= 8   Permanent employees= 6  Contract employees= 2


Employee 3: Emily Willow
Employee 3: Willow Emily
Employee 4: Jhon Reid
Employee 4: Reid Jhon

Employee destructor for James Wangis called
Employee destructor for Tomas Hwangis called
Employee destructor for Emily Willowis called
Employee destructor for Robert Jonesis called
Employee destructor for Susan Bakeris called
Employee destructor for RB Linis called
Employee destructor for Vincent Urlis called
Employee destructor for Maria Vinciis called
Employee destructor for Jhon Reidis called

After objects are deleted, Total number of employees= 0   Permanent employees= 0   Contract employees= 0
```

```
// Fig. 10.20: Employee.h
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include <string>
using namespace std;

class Employee
{
public:
   Employee( const string &, const string & ); // constructor
   ~Employee(); // destructor
   string getFirstName() const; // return first name
   string getLastName() const; // return last name
   // static member function
   static int getCount(); // return number of objects instantiated
private:
   string firstName;
   string lastName;
   // static data
   static int count; // number of objects instantiated
}; // end class Employee
#endif
```

```
// Fig. 10.21: Employee.cpp
// Employee class member-function definitions.
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

// define and initialize static data member at global namespace scope
int Employee::count = 0; // cannot include keyword static

// define static member function that returns number of
// Employee objects instantiated (declared static in Employee.h)
int Employee::getCount()
{
   return count;
} // end static function getCount
```

```
// constructor initializes non-static data members and
// increments static data member count
Employee::Employee( const string &first, const string &last )
   : firstName( first ), lastName( last )
{
   ++count; // increment static count of employees
   cout << "Employee constructor for " << firstName
      << ' ' << lastName << " called." << endl;
} // end Employee constructor

// destructor deallocates dynamically allocated memory
Employee::~Employee()
{
   cout << "~Employee() called for " << firstName
      << ' ' << lastName << endl;
   --count; // decrement static count of employees
} // end ~Employee destructor
```

```
// return first name of employee
string Employee::getFirstName() const
{
   return firstName; // return copy of first name
} // end function getFirstName

// return last name of employee
string Employee::getLastName() const
{
   return lastName; // return copy of last name
} // end function getLastName
```