Module 16: Programming in C++

static Members

Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

sourangshu@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in C++

by **Prof. Partha Pratim Das**

Module Objectives

- Understand `static` data member and member function

Module Outline

- `static` data member
  - Print Task
- `static` member function
  - Print Task
- Singleton Class

`static` Data Member

A `static` data member

- is associated with class not with object
- is shared by all the objects of a class
- needs to be defined outside the class scope (in addition to the declaration within the class scope) to avoid linker error
- must be initialized in a source file
- is constructed before `main()` starts and destructed after `main()` ends
- can be private / public type
- can be accessed
  - with the class-name followed by the scope resolution operator (::)
  - as a member of any object of the class
- virtually eliminates any need for global variables in OOPs environment

Program 16.01: static Data Member A Simple Case

Non static Data Member

```
#include<iostream>
using namespace std;
class MyClass { int x; // Non-static
public:
    void set() { x = 15; }
    void print() {
        x = x + 10;
        cout << "x =" << x << endl ;
    }
};

int main() {
    MyClass obj1, obj2;
    obj1.set(); obj2.set();

    obj1.print(); obj2.print();
    return 0 ;
}
---
x = 25 , x = 25
```

- x is a non-static data member
- x cannot be shared between obj1 & obj2

- Non-static data members do not need separate definitions - instantiated with the object
- Non-static data members are initialized during object construction

static Data Member

```
#include<iostream>
using namespace std;
class MyClass { static int x; // Declare static
public:
    void set() { x = 15; }
    void print() {
        x = x + 10;
        cout << "x =" << x << endl;
    }
};
int MyClass::x = 0; // Define static data member
int main() {
    MyClass obj1, obj2;
    obj1.set(); obj2.set();

    obj1.print(); obj2.print();
    return 0;
}
---
x = 25 , x = 35
```

- x is static data member
- x is shared by all MyClass objects including obj1 & obj2
- static data members must be defined in the global scope
- static data members are initialized during program start-up

Program 16.02: static Data Member Print Task


```
#include <iostream>
using namespace std;
class PrintJobs { int nPages_;  // # of pages in current job
public:
    static int nTrayPages_;     // # of pages remaining in the tray
    static int nJobs_;          // # of print jobs executing
    PrintJobs(int nP): nPages_(nP) {
        ++nJobs_;
        cout << "Printing " << nP << " pages" << endl;
        nTrayPages_ = nTrayPages_ - nP;
    }
    ~PrintJobs() { --nJobs_; }
};
int PrintJobs::nTrayPages_ = 500; // Definition and initialization -- load paper
int PrintJobs::nJobs_ = 0;        // Definition and initialization -- no job to start with
int main() {
    cout << "Jobs = " << PrintJobs::nJobs_ << endl;
    cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
    PrintJobs job1(10);
    cout << "Jobs = " << PrintJobs::nJobs_ << endl;
    cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
    {
        PrintJobs job1(30), job2(20);
        cout << "Jobs = " << PrintJobs::nJobs_ << endl;
        cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
        PrintJobs::nTrayPages_ += 100; // Load 100 more pages
    }
    cout << "Jobs = " << PrintJobs::nJobs_ << endl;
    cout << "Pages= " << PrintJobs::nTrayPages_ << endl;
    return 0;
}
```

```
Output:

Jobs = 0
Pages= 500
Printing 10 pages
Jobs = 1
Pages= 490
Printing 30 pages
Printing 20 pages
Jobs = 3
Pages= 440
Jobs = 1
Pages= 540
```

`static` Member Function

A `static` member function

- does not have `this` pointer – not associated with any object
- cannot access non-static data members
- cannot invoke non-static member functions
- can be accessed
  - with the class-name followed by the scope resolution operator (::)
  - as a member of any object of the class
- is needed to read / write static data members
  - Again, for encapsulation static data members should be private
  - get()-set() idiom is built for access (static member functions in public)
- may initialize static data members even before any object creation
- cannot co-exist with a non-static version of the same function
- cannot be declared as `const`

Program 16.03: static Data & Member Function Print Task (safe)


```
#include <iostream>
using namespace std;
class PrintJobs { int nPages_; // # of pages in current job
    static int nTrayPages_;    // # of pages remaining in the tray
    static int nJobs_;         // # of print jobs executing
public: PrintJobs(int nP) : nPages_(nP) { ++nJobs_;
            cout << "Printing " << nP << " pages" << endl;
            nTrayPages_ = nTrayPages_ - nP;
        }
    ~PrintJobs() { --nJobs_; }
    static int getJobs() { return nJobs_; }
    static int checkPages() { return nTrayPages_; }
    static void loadPages(int nP) { nTrayPages_ += nP; }
};
int PrintJobs::nTrayPages_ = 500; // Definition and initialization -- load paper
int PrintJobs::nJobs_ = 0;        // Definition and initialization -- no job to start with
int main() {
    cout << "Jobs = " << PrintJobs::getJobs() << endl;
    cout << "Pages= " << PrintJobs::checkPages() << endl;
    PrintJobs job1(10);
    cout << "Jobs = " << PrintJobs::getJobs() << endl;
    cout << "Pages= " << PrintJobs::checkPages() << endl;
    {   PrintJobs job1(30), job2(20);
        cout << "Jobs = " << PrintJobs::getJobs() << endl;
        cout << "Pages= " << PrintJobs::checkPages() << endl;
        PrintJobs::loadPages(100);
    }
    cout << "Jobs = " << PrintJobs::getJobs() << endl;
    cout << "Pages= " << PrintJobs::checkPages() << endl;
    return 0;
}
```

```
Output:

Jobs = 0
Pages= 500
Printing 10 pages
Jobs = 1
Pages= 490
Printing 30 pages
Printing 20 pages
Jobs = 3
Pages= 440
Jobs = 1
Pages= 540
```

Singleton Class

- A class is called a Singleton if it can have *only* one instance
- Many classes are singleton:
  - President of India
  - Prime Minister of India
  - Director of IIT Kharagpur
  - ...
- How to implement a Singleton Class?
- How to restrict that user can created *only one* instance?

Program 16.04: static Data & Member Function Singleton Printer


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

class Printer {           /* THIS IS A SINGLETON PRINTER -- ONLY ONE INSTANCE */
    bool blackAndWhite_, bothSided_;

    Printer(bool bw = false, bool bs = false) : blackAndWhite_(bw), bothSided_(bs)
    { cout << "Printer constructed" << endl; } // Private -- Printer cannot be constructed!

    static Printer *myPrinter_; // Pointer to the Instance of the Singleton Printer

public:
    ~Printer() { cout << "Printer destructed" << endl; }

    static const Printer& printer(bool bw = false, bool bs = false) { // Access the Printer
        if (!myPrinter_) myPrinter_ = new Printer(bw, bs); // Constructed for first call
        return *myPrinter_;                                // Reused from next time
    }
    void print(int nP) const { cout << "Printing " << nP << " pages" << endl; }
};
Printer *Printer::myPrinter_ = 0;
int main() {
    Printer::printer().print(10);
    Printer::printer().print(20);

    delete &Printer::printer();
    return 0;
}
```

```
Output:

Printer constructed
Printing 10 pages
Printing 20 pages
Printer destructed
```

In the recorded video the destructor was directly called by **Printer::printer().~Printer();** *This is wrong and will leak memory. It is corrected here with* **delete &Printer::printer();**

Program 16.05: Using function-local static Data Singleton Printer


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

class Printer {                 /* THIS IS A SINGLETON PRINTER -- ONLY ONE INSTANCE */
    bool blackAndWhite_, bothSided_;

    Printer(bool bw = false, bool bs = false) : blackAndWhite_(bw), bothSided_(bs)
    { cout << "Printer constructed" << endl; }

    ~Printer() { cout << "Printer destructed" << endl; }
public:

    static const Printer& printer(bool bw = false, bool bs = false) {
        static Printer myPrinter(bw, bs); // The Singleton -- constructed the first time

        return myPrinter;
    }
    void print(int nP) const {
        cout << "Printing " << nP << " pages" << endl;
    }
};
int main() {
    Printer::printer().print(10);
    Printer::printer().print(20);

    return 0;
}
```

```
Output:

Printer constructed
Printing 10 pages
Printing 20 pages
Printer destructed
```

- Function local `static` object is used
- No memory management overhead – so destructor too get `private`
- This is called *Meyer's Singleton*

Module Summary

- Introduced `static` data member
- Introduced `static` member function
- Exposed to use of `static` members
- Singleton Class discussed