Module 15: Programming in C++

Const-ness

Intructors: Abir Das and Sourangshu Bhattacharya

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

{*abir, sourangshu*}*@cse.iitkgp.ac.in*

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**

Module Objectives

- Understand const-ness of objects in C++
- Understand the use of const-ness in class design

Module Outline

Constant Objects

- Like objects of built-in type, objects of user-defined types can also be made constant
- If an object is constant, none of its data members can be changed
- The type of the this pointer of a constant object of class, say, MyClass is:

```
// const Pointer to const Object
const MyClass * const this;
```

instead of

```
// const Pointer to non-const Object
MyClass * const this;
```

as for a non-constant object of the same class

- A constant object cannot invoke normal methods of the class as these methods can change the object

Program 15.01: Non-Constant Objects

```
#include <iostream>
using namespace std;
class MyClass { int myPriMember_;
public: int myPubMember_;
    MyClass(int mPri, int mPub) : myPriMember_(mPri), myPubMember_(mPub) { }
    int getMember() { return myPriMember_; }
    void setMember(int i) { myPriMember_ = i; }
    void print() { cout << myPriMember_ << ", " << myPubMember_ << endl; }
};
int main() { MyClass myObj(0, 1);            // Non-constant object

    cout << myObj.getMember() << endl;
    myObj.setMember(2);
    myObj.myPubMember_ = 3;
    myObj.print();
}
---
0
2, 3
```

- It is okay to invoke methods for non-constant object **myObj**
- It is okay to make changes in non-constant object **myObj** by method (**setMember()**)
- It is okay to make changes in non-constant object **myObj** directly (**myPubMember_**)

Program 15.02: Constant Objects

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

class MyClass { int myPriMember_; public: int myPubMember_;
    MyClass(int mPri, int mPub) : myPriMember_(mPri), myPubMember_(mPub) { }
    int getMember() { return myPriMember_; }
    void setMember(int i) { myPriMember_ = i; }
    void print() { cout << myPriMember_ << ", " << myPubMember_ << endl; }
};
int main() { const MyClass myConstObj(5, 6); // Constant object

    cout << myConstObj.getMember() << endl; // Error 1
    myConstObj.setMember(7);                // Error 2
    myConstObj.myPubMember_ = 8;            // Error 3
    myConstObj.print();                     // Error 4
}
```

- It is not allowed to invoke methods or make changes in constant object **myConstObj**
- Error (1, 2 & 4) on method invocation typically is:
  cannot convert 'this' pointer from 'const MyClass' to 'MyClass &'
- Error (3) on member update typically is:
  'myConstObj' : you cannot assign to a variable that is const
- With const, **this** pointer is **const MyClass * const** while the methods expects **MyClass * const**
- Consequently, we cannot print the data member of the class (even without changing it)
- Fortunately, constant objects can invoke (select) methods if they are **constant member functions**

Constant Member Function

- To declare a constant member function, we use the keyword `const` between the function header and the body. Like:

```
void print() const { cout << myMember_ << endl; }
```

- A constant member function expects a `this` pointer as:

```
const MyClass * const this;
```

  and hence can be invoked by constant objects
- In a constant member function no data member can be changed. Hence,

```
void setMember(int i) const
{ myMember_ = i; } // data member cannot be changed
```

  gives an error
- Interesting, *non-constant objects* can invoke *constant member functions* (by casting – we discuss later) and, of course, *non-constant member functions*
- *Constant objects*, however, can **only** invoke *constant member functions*
- **All member functions that do not need to change an object must be declared as constant member functions**

Program 15.03: Constant Member Functions

```
#include <iostream>
using namespace std;
class MyClass { int myPriMember_; public: int myPubMember_;
    MyClass(int mPri, int mPub) : myPriMember_(mPri), myPubMember_(mPub) { }
    int getMember() const { return myPriMember_; }                     // const Member Func.
    void setMember(int i) { myPriMember_ = i; }                        // non-const Member Func.
    void print() const { cout << myPriMember_ << ", " << myPubMember_ << endl; } // const Member Func.
};
int main() { MyClass myObj(0, 1);   // non-const object
    const MyClass myConstObj(5, 6); // const object
    // non-const object can invoke all member functions and update data members
    cout << myObj.getMember() << endl;
    myObj.setMember(2);
    myObj.myPubMember_ = 3;
    myObj.print();
    // const object cannot allow any change
    cout << myConstObj.getMember() << endl;
    // myConstObj.setMember(7);     // Cannot invoke non-const member functions
    // myConstObj.myPubMember_ = 8; // Cannot update data member
    myConstObj.print();
}
```

Output

```
0
2, 3
5
5, 6
```

- Now **myConstObj** can invoke **getMember()** and **print()**, but cannot invoke **setMember()**
- Naturally **myConstObj** cannot update **myPubMember_**
- **myObj** can invoke all of **getMember()**, **print()**, and **setMember()**

Constant Data members

- Often we need part of an object, that is, one or more data members to be constant (non-changeable after construction) while the rest of the data members should be changeable. For example:
  - For an **Employee**: `employee ID` and `DoB` should be *non-changeable* while `designation`, `address`, `salary` etc. should be *changeable*
  - For a **Student**: `roll number` and `DoB` should be *non-changeable* while `year of study`, `address`, `gpa` etc. should be *changeable*
  - For a **Credit Card**[1]: `card number` and `name of holder` should be *non-changeable* while `date of issue`, `date of expiry`, `address`, `cvv number` etc. should be *changeable*
- We do this by making the *non-changeable* data members as constant by putting the `const` keyword before the declaration of the member in the class
- **A constant data member cannot be changed even in a non-constant object**
- **A constant data member must be initialized on the initialization list**

[1] May not hold for a card that changes number on re-issue

Program 15.04: Constant Data Member

```
#include <iostream>
using namespace std;
class MyClass { const int cPriMem_; /* const data member */ int priMem_; public:
    const int cPubMem_; /* const data member */ int pubMem_;
    MyClass(int cPri, int ncPri, int cPub, int ncPub) :
        cPriMem_(cPri), priMem_(ncPri), cPubMem_(cPub), pubMem_(ncPub) { }
    int getcPri() { return cPriMem_;  }
    void setcPri(int i) { cPriMem_ = i; } // Error 1: Assignment to const data member
    int getPri() { return priMem_; }
    void setPri(int i) { priMem_ = i; }
};
int main() { MyClass myObj(1, 2, 3, 4);

    cout << myObj.getcPri() << endl; myObj.setcPri(6);
    cout << myObj.getPri() << endl; myObj.setPri(6);

    cout << myObj.cPubMem_ << endl;
    myObj.cPubMem_ = 3;                     // Error 2: Assignment to const data member

    cout << myObj.pubMem_ << endl; myObj.pubMem_ = 3;
}
```

- It is not allowed to make changes to constant data members in **myObj**
- Error 1: **l-value specifies const object**
- Error 2: 'myObj' : you cannot assign to a variable that is const

Credit Card Example

We now illustrate constant data members with a complete example of `CreditCard` class with the following supporting classes:

- `String` class
- `Date` class
- `Name` class
- `Address` class

Program 15.05: String Class: String.h

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

class String { char *str_; size_t len_;
public:
    String(const char *s) : str_(strdup(s)), len_(strlen(str_))          // Ctor
    { cout << "String ctor: "; print(); cout << endl; }
    String(const String& s) : str_(strdup(s.str_)), len_(strlen(str_))     // CCtor
    { cout << "String cctor: "; print(); cout << endl; }
    String& operator=(const String& s) {
        if (this != &s) {
            free(str_);
            str_ = strdup(s.str_);
            len_ = s.len_;
        }
        return *this;
    }
    ~String() { cout << "String dtor: "; print(); cout << endl; free(str_); } // Dtor
    void print() const { cout << str_; }
};
```

- Copy Constructor and Copy Assignment Operator added
- `print()` made a constant member function

Program 15.05: Date Class: Date.h

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

char monthNames[][4]={ "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
char dayNames[][10]={ "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
class Date {
    enum Month { Jan = 1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
    enum Day { Mon, Tue, Wed, Thr, Fri, Sat, Sun };
    typedef unsigned int UINT;
    UINT date_; Month month_; UINT year_;
public:
    Date(UINT d, UINT m, UINT y) : date_(d), month_((Month)m), year_(y)
    { cout << "Date ctor: "; print(); cout << endl; }
    Date(const Date& d) : date_(d.date_), month_(d.month_), year_(d.year_)
    { cout << "Date cctor: "; print(); cout << endl; }
    Date& operator=(const Date& d) { date_ = d.date_; month_ = d.month_; year_ = d.year_; return *this; }
    ~Date() { cout << "Date dtor: "; print(); cout << endl; }
    void print() const { cout << date_ << "/" << monthNames[month_ - 1] << "/" << year_; }
    bool validDate() const { /* Check validity */ return true; }             // Not Implemented
    Day day() const { /* Compute day from date using time.h */ return Mon; } // Not Implemented
};
```

- Copy Constructor and Copy Assignment Operator added
- `print()`, `validDate()`, and `day()` made constant member functions

Program 15.05: Name Class: Name.h

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

#include "String.h"

class Name { String firstName_, lastName_;
public:
    Name(const char* fn, const char* ln) : firstName_(fn), lastName_(ln)    // Uses Ctor of String class
    { cout << "Name ctor: "; print(); cout << endl; }
    Name(const Name& n) : firstName_(n.firstName_), lastName_(n.firstName_) // Uses CCtor of String class
    { cout << "Name cctor: "; print(); cout << endl; }
    Name& operator=(const Name& n) {
        firstName_ = n.firstName_; // Uses operator=() of String class
        lastName_ = n.lastName_;   // Uses operator=() of String class
        return *this;
    }
    ~Name() { cout << "Name dtor: "; print(); cout << endl; } // Uses Dtor of String class
    void print() const // Uses print() of String class
    { firstName_.print(); cout << " "; lastName_.print(); }
};
```

- Copy Constructor and Copy Assignment Operator added
- `print()` made a constant member function

Program 15.05: Address Class: Address.h

```
#include <iostream>
using namespace std;
#include "String.h"

class Address { unsigned int houseNo_; String street_, city_, pin_;
public:
    Address(unsigned int hn, const char* sn, const char* cn, const char* pin): // Uses Ctor of String class
        houseNo_(hn), street_(sn), city_(cn), pin_(pin)
    { cout << "Address ctor: "; print(); cout << endl; }
    Address(const Address& a): // Uses CCtor of String class
        houseNo_(a.houseNo_), street_(a.street_), city_(a.city_), pin_(a.pin_)
    { cout << "Address cctor: "; print(); cout << endl; }
    Address& operator=(const Address& a) { // Uses operator=() of String class
        houseNo_ = a.houseNo_; street_ = a.street_; city_ = a.city_; pin_ = a.pin_; return *this; }
    ~Address() { cout << "Address dtor: "; print(); cout << endl; } // Uses Dtor of String class
    void print() const { // Uses print() of String class
        cout << houseNo_ << " "; street_.print(); cout << " ";
        city_.print(); cout << " "; pin_.print();
    }
};
```

- Copy Constructor and Copy Assignment Operator added
- print() made a constant member function

Program 15.05: Credit Card Class: CreditCard.h

```
#include <iostream>
using namespace std;
#include "Date.h"
#include "Name.h"
#include "Address.h"
class CreditCard { typedef unsigned int UINT; char *cardNumber_;
    Name holder_; Address addr_; Date issueDate_, expiryDate_; UINT cvv_;
public: CreditCard(const char* cNumber, const char* fn, const char* ln, unsigned int hn, const char* sn,
    const char* cn, const char* pin, UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear,
    UINT cvv): holder_(fn, ln), addr_(hn, sn, cn, pin), issueDate_(1, issueMonth, issueYear),
    expiryDate_(1, expiryMonth, expiryYear), cvv_(cvv) // Uses Ctor's of Date, Name, Address
    { cardNumber_ = new char[strlen(cNumber) + 1]; strcpy(cardNumber_, cNumber);
          cout << "CC ctor: "; print(); cout << endl; }
    // Uses Dtor's of Date, Name, Address
    ~CreditCard() { cout << "CC dtor: "; print(); cout << endl; delete[] cardNumber_; }
    void setHolder(const Name& h)      { holder_ = h; }      // Change holder name
    void setAddress(const Address& a) { addr_ = a; }        // Change address
    void setIssueDate(const Date& d)  { issueDate_ = d; }  // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; } // Change expiry date
    void setCVV(UINT v)               { cvv_ = v; }         // Change cvv number
    void print() const { cout<<cardNumber_<<" "; holder_.print(); cout<<" "; addr_.print();
        cout<<" "; issueDate_.print(); cout<<" "; expiryDate_.print(); cout<<" "; cout<<cvv_; }
};
```

- Set methods added
- `print()` made a constant member function

Program 15.05: Credit Card Class Application

```
#include <iostream>
using namespace std;
#include "CreditCard.h"

int main() { CreditCard cc("5321711934640027", "Sherlock", "Holmes",
                          221, "Baker Street", "London", "NW1 6XE", 7, 2014, 6, 2016, 811);
    cout << endl; cc.print(); cout << endl << endl;;

    cc.setHolder(Name("David", "Cameron"));
    cc.setAddress(Address(10, "Downing Street", "London", "SW1A 2AA"));
    cc.setIssueDate(Date(1, 7, 2017));
    cc.setExpiryDate(Date(1, 6, 2019));
    cc.setCVV(127);
    cout << endl; cc.print(); cout << endl << endl;;
}
```

```
// Construction of Data Members & Object
5321711934640027 Sherlock Holmes 221 Baker Street London NW1 6XE 1/Jul/2014 1/Jun/2016 811

// Construction & Destruction of temporary objects
5321711934640027 David Cameron 10 Downing Street London SW1A 2AA 1/Jul/2017 1/Jun/2019 127

// Destruction of Data Members & Object
```

- We could change address, issue date, expiry date, and cvv. This is fine
- **We could change the name of the holder! This should not be allowed**

Program 15.06: Credit Card Class: Constant data members

```
// Include <iostream>, "String.h", "Date.h", "Name.h", "Address.h"
using namespace std;

class CreditCard { typedef unsigned int UINT;
    char *cardNumber_;
    const Name holder_;          // Holder name cannot be changed after construction
    Address addr_;  Date issueDate_, expiryDate_; UINT cvv_;
public: CreditCard(...) : ... { ... } ~CreditCard() { ... }

    void setHolder(const Name& h)     { holder_ = h; }      // Change holder name
    // error C2678: binary '=' : no operator found which takes a left-hand operand
    // of type 'const Name' (or there is no acceptable conversion)

    void setAddress(const Address& a) { addr_ = a; }        // Change address
    void setIssueDate(const Date& d)  { issueDate_ = d; }   // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; }  // Change expiry date
    void setCVV(UINT v)               { cvv_ = v; }         // Change cvv number

    void print() { ... }
};
```

- We prefix `Name holder_` with `const`. Now the holder name cannot be changed after construction
- **In `setHolder()`, we get a compilation error for `holder_ = h;` in an attempt to change `holder_`**
- With `const` prefix `Name holder_` becomes constant – unchangeable

Program 15.06: Credit Card Class: Clean

```
// Include <iostream>, "String.h", "Date.h", "Name.h", "Address.h"
using namespace std;

class CreditCard { typedef unsigned int UINT;
    char *cardNumber_;
    const Name holder_;         // Holder name cannot be changed after construction
    Address addr_;
    Date issueDate_, expiryDate_; UINT cvv_;
public:
    CreditCard(...) : ... { ... }
    ~CreditCard() { ... }

    void setAddress(const Address& a)  addr_ = a;         // Change address
    void setIssueDate(const Date& d)   issueDate_ = d;    // Change issue date
    void setExpiryDate(const Date& d)  expiryDate_ = d;   // Change expiry date
    void setCVV(UINT v)                cvv_ = v;          // Change cvv number

    void print() { ... }
};
```

- Method `setHolder()` removed

Program 15.06: Credit Card Class Application: Revised

```
#include <iostream>
using namespace std;
#include "CreditCard.h"
int main() {
    CreditCard cc("5321711934640027", "Sherlock", "Holmes",
                  221, "Baker Street", "London", "NW1 6XE", 7, 2014, 6, 2016, 811);
    cout << endl; cc.print(); cout << endl << endl;;

//    cc.setHolder(Name("David", "Cameron"));
    cc.setAddress(Address(10, "Downing Street", "London", "SW1A 2AA"));
    cc.setIssueDate(Date(1, 7, 2017));
    cc.setExpiryDate(Date(1, 6, 2019));
    cc.setCVV(127);
    cout << endl; cc.print(); cout << endl << endl;;
}
```

```
// Construction of Data Members & Object
5321711934640027 Sherlock Holmes 221 Baker Street London NW1 6XE 1/Jul/2014 1/Jun/2016 811

// Construction & Destruction of temporary objects
5321711934640027 Sherlock Holmes 10 Downing Street London SW1A 2AA 1/Jul/2017 1/Jun/2019 127

// Destruction of Data Members & Object
```

- Now holder_ cannot be changed. So we are safe
- **However, it is still possible to replace or edit the card number. This, too, should be disallowed**

Program 15.07: Credit Card Class: cardNumber_ Issue

```
// Include <iostream>, "String.h", "Date.h", "Name.h", "Address.h"
using namespace std;

class CreditCard { typedef unsigned int UINT;
    char *cardNumber_;          // Card number is editable as well as replaceable
    const Name holder_;         // Holder name cannot be changed after construction
    Address addr_;
    Date issueDate_, expiryDate_; UINT cvv_;
public:
    CreditCard(...) : ... { ... }
    ~CreditCard() { ... }

    void setAddress(const Address& a) { addr_ = a; }        // Change address
    void setIssueDate(const Date& d)  { issueDate_ = d; }  // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; } // Change expiry date
    void setCVV(UINT v)               { cvv_ = v; }        // Change cvv number

    void print() { ... }
};
```

- It is still possible to replace or edit the card number
- To make the cardNumber_ *non-replaceable*, we need to make this *constant pointer*
- Further, to make it *non-editable* we need to make cardNumber_ point to a *constant string*
- Hence, we change char *cardNumber_ to const char * const cardNumber_

Program 15.07: Credit Card Class: cardNumber_ Issue

```
// Include <iostream>, "String.h", "Date.h", "Name.h", "Address.h"
using namespace std;
class CreditCard {
    typedef unsigned int UINT;
    const char * const cardNumber_; // Card number cannot be changed after construction
    const Name holder_;             // Holder name cannot be changed after construction
    Address addr_; Date issueDate_, expiryDate_; UINT cvv_;
public: CreditCard(const char* cNumber, const char* fn, const char* ln,
        unsigned int hn, const char* sn, const char* cn, const char* pin,
        UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear, UINT cvv) :
        holder_(fn, ln), addr_(hn, sn, cn, pin), issueDate_(1, issueMonth, issueYear),
        expiryDate_(1, expiryMonth, expiryYear), cvv_(cvv) {
        cardNumber_ = new char[strlen(cNumber) + 1]; // ERROR: No assignment to const pointer
        strcpy(cardNumber_, cNumber);                // ERROR: No copy to const C-string
        cout << "CC ctor: "; print(); cout << endl;
    }
    ~CreditCard() { cout << "CC dtor: "; print(); cout << endl; delete[] cardNumber_; }

    // Set methods and print method skipped ...
};
```

- cardNumber_ is now a *constant pointer to a constant string*
- With this the allocation for the C-string fails in the body as constant pointer cannot be assigned
- Further, copy of C-string (strcpy()) fails as copy of constant C-string is not allowed
- **We need to move these codes to the initialization list**

Program 15.07: Credit Card Class: cardNumber_ Issue: Resolved

```
// Include <iostream>, "String.h", "Date.h", "Name.h", "Address.h"
using namespace std;
class CreditCard { typedef unsigned int UINT;
    const char * const cardNumber_; // Card number cannot be changed after construction
    const Name holder_;             // Holder name cannot be changed after construction
    Address addr_; Date issueDate_, expiryDate_; UINT cvv_;
public: CreditCard(const char* cNumber, const char* fn, const char* ln,
        unsigned int hn, const char* sn, const char* cn, const char* pin,
        UINT issueMonth, UINT issueYear, UINT expiryMonth, UINT expiryYear, UINT cvv) :
        cardNumber_(strcpy(new char[strlen(cNumber)+1], cNumber)),
        holder_(fn, ln), addr_(hn, sn, cn, pin), issueDate_(1, issueMonth, issueYear),
        expiryDate_(1, expiryMonth, expiryYear), cvv_(cvv)
    { cout << "CC ctor: "; print(); cout << endl; }
    ~CreditCard() { cout << "CC dtor: "; print(); cout << endl; delete[] cardNumber_; }
    void setAddress(const Address& a) { addr_ = a; }        // Change address
    void setIssueDate(const Date& d)  { issueDate_ = d; }   // Change issue date
    void setExpiryDate(const Date& d) { expiryDate_ = d; } // Change expiry date
    void setCVV(UINT v)               { cvv_ = v; }         // Change cvv number
    void print() const { cout<<cardNumber_<<" "; holder_.print(); cout<<" "; addr_.print();
        cout<<" "; issueDate_.print(); cout<<" "; expiryDate_.print(); cout<<" "; cout<<cvv_; }
};
```

- Note the initialization of cardNumber_ in initialization list
- **All constant data members must be initialized in initialization list**

`mutable` Members

`mutable` Members

`mutable` Data Members

- While a *constant* data member is *not changeable* even in a *non-constant object*, a **mutable** data member is *changeable* in a *constant object*
- `mutable` is provided to model *Logical (Semantic) const-ness* against the default *Bit-wise (Syntactic) const-ness* of C++
- Note that:
  - `mutable` is applicable only to data members and not to variables
  - Reference data members cannot be declared `mutable`
  - Static data members cannot be declared `mutable`
  - `const` data members cannot be declared `mutable`
- If a data member is declared `mutable`, then it is legal to assign a value to it from a `const` member function

Program 15.08: mutable Data Members

```
#include <iostream>
using namespace std;
class MyClass {
    int mem_;
    mutable int mutableMem_;
public:
    MyClass(int m, int mm) : mem_(m), mutableMem_(mm) { }
    int getMem() const { return mem_; }
    void setMem(int i) { mem_ = i; }
    int getMutableMem() const { return mutableMem_; }
    void setMutableMem(int i) const { mutableMem_ = i; } // Okay to change mutable
};
int main() { const MyClass myConstObj(1, 2);

    cout << myConstObj.getMem() << endl;
    // myConstObj.setMem(3);                // Error to invoke

    cout << myConstObj.getMutableMem() << endl;
    myConstObj.setMutableMem(4);
}
```

- **setMutableMem()** is a constant member function so that constant **myConstObj** can invoke it
- **setMutableMem()** can still set **mutableMem_** because **mutableMem_** is **mutable**
- In contrast, **myConstObj** cannot invoke **setMem()** and hence **mem_** cannot be changed

Logical vis-a-vis Bit-wise Const-ness

- `const` in C++, models *bit-wise* constant. Once an object is declared `const`, no part (actually, *no bit*) of it can be changed after construction (and initialization)
- However, while programming we often need an object to be *logically* constant. That is, the concept represented by the object should be constant; but if its representation need more data members for computation and modeling, these have no reason to be constant.
- `mutable` allows such surrogate data members to be changeable in a (bit-wise) constant object to model logically const objects
- To use `mutable` we shall look for:
  - A logically constant concept
  - A need for data members outside the representation of the concept; but are needed for computation

Program 15.09: When to use mutable Data Members?

- Typically, when a class represents a constant concept, and
- It computes a value first time and caches the result for future use

```
// Source: http://www.highprogrammer.com/alan/rants/mutable.html
#include <iostream>
using namespace std;
class MathObject {                          // Constant concept of PI
    mutable bool piCached_;                 // Needed for computation
    mutable double pi_;                     // Needed for computation
public:
    MathObject() : piCached_(false) { } // Not available at construction
    double pi() const {                     // Can access PI only through this method
        if (!piCached_) {                   // An insanely slow way to calculate pi
            pi_ = 4;
            for (long step = 3; step < 1000000000; step += 4) {
                pi_ += ((-4.0 / (double)step) + (4.0 / ((double)step + 2)));
            }
            piCached_ = true;               // Now computed and cached
        }
        return pi_;
    }
};
int main() { const MathObject mo; cout << mo.pi() << endl; /* Access PI */ }
```

- Here a MathObject is logically constant; but we use mutable members for computation

Program 15.10: When *not* to use mutable Data Members?

- mutable should be **rarely used** – only when it is really needed. A bad example follows:

Improper Design (mutable)	Proper Design (const)

```
class Employee { string _name, _id;
    mutable double _salary;
public: Employee(string name = "No Name",
        string id = "000-00-0000",
        double salary = 0): _name(name), _id(id)
    { _salary = salary; }
    string getName() const;
    void setName(string name);
    string getid() const;
    void setid(string id);
    double getSalary() const;
    void setSalary(double salary);
    void promote(double salary) const
    { _salary = salary; }
};
---
const Employee john("JOHN","007",5000.0);
// ...
john.promote(20000.0);
```
	```
	class Employee { const string _name, _id;
	    double _salary;
	public: Employee(string name = "No Name",
	        string id = "000-00-0000",
	        double salary = 0): _name(name), _id(id)
	    { _salary = salary; }
	    string getName() const;
	    // void setName(string name); // _name is const
	    string getid() const;
	    // void setid(string id);     // _id is const
	    double getSalary() const;
	    void setSalary(double salary);
	    void promote(double salary)
	    { _salary = salary; }
	};
	---
	Employee john("JOHN","007",5000.0);
	// ...
	john.promote(20000.0);
	```

- Employee is not logically constant. If it is, then _salary should also be const
- Design on right makes that explicit