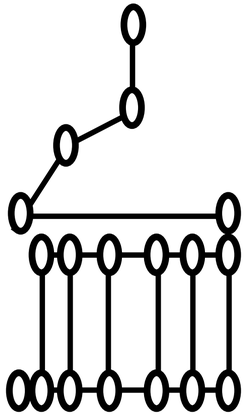
Lecture: Composition & Constant Members

ENGR 2730: Computers in Engineering



Constant data members

Constant data members

*Use the keyword **const** to specify that an object is not modifiable and that any attempt to modify an object should result in a compilation error.*

requires that you ONLY use constant member functions with the object

Side note: we have already seen **const** member functions (used to indicate that a member function will not modify its data members)

```
void print() const;
```

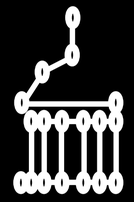
Constant data members

*Use the keyword **const** to specify that an object is not modifiable and that any attempt to modify an object should result in a compilation error.*

requires that you ONLY use constant member functions with the object

But how do we "initialize" constant data members if we can only call constant member functions with the object?

```
void print() const;
```



Constant data members

*Use the keyword **const** to specify that an object is not modifiable and that any attempt to modify an object should result in a compilation error.*

requires that you ONLY use constant member functions with the object

But how do we "initialize" constant data members if we can only call constant member functions with the object?

Use a **member initializer list** with the constructor.

Composition

Composition (*has-a* relationship)

With composition, a class has objects of other classes as data members.

Analogous to real-life objects:

A car has four tires, a body, an engine, a radio, ...

A polynomial has a set of terms...

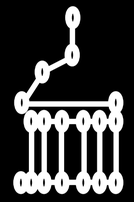
A classroom has desks, a computer, a projector, a teacher, students...

Case study: The Employee class

```
class Employee
{
public:
    Employee( const string &first, const string &last,
              const Date &dateOfBirth, const Date &dateOfHire );
    void print() const;

private:
    string firstName;      // composition: member object
    string lastName;       // composition: member object
    const Date birthDate;  // composition: member object
    const Date hireDate;   // composition: member object
};
```

Employee.h



Case study: The Employee class

```
class Employee
{
public:
     Employee( const string &first, const string &last,
               const Date &dateOfBirth, const Date &dateOfHire );
     void print() const;

private:
     string firstName;       // composition: member object
     string lastName;        // composition: member object
     const Date birthDate;   // composition: member object
     const Date hireDate;    // composition: member object
};
```

custom Date class

Employee.h

The Date class

monthsPerYear has the same value for all instantiated objects.

Share a single classwide static copy of monthsPerYear.

```
class Date
{
public:
    static const int monthsPerYear = 12; // number of months in a year
    Date (int mn = 1, int dy = 1, int yr = 1900); // constructors
    void print() const; // print date in month/day/year format

private:
    int month; // 1-12 (January-December)
    int day;   // 1-31 based on month
    int year;  // any year

    // utility function to check if day is proper for month and year
    int checkDay( int testDay ) const;
};
```

Date.h

Static variables & Member functions

- static data members have ONE copy shared between ALL instance objects of a class
- static function local variables have ONE copy shared for all calls of the member function
- static variables are often good for constants that are required to be the same for all objects
- static variables are often good when their value takes a long time to compute, but then never changes

Performance Tip 9.5

Use `static` *data members to save storage when a single copy of the data for all objects of a class will suffice—such as a constant that can be shared by all objects of the class.*

Date member functions

```
// constructor confirms proper value for month; calls
// utility function checkDay to confirm proper value for day
Date::Date( int mn, int dy, int yr )
{
   if ( mn > 0 && mn <= monthsPerYear ) // validate the month
     month = mn;
   else
   {
      month = 1; // invalid month set to 1
      cout << "Invalid month (" << mn << ") set to 1.\n";
   }

   year = yr; // could validate yr
   day = checkDay( dy ); // validate the day

   // output Date object to show when its constructor is called
   cout << "Date object constructor for date ";
   print();
   cout << endl;
}
```

Date.cpp

Date member functions

This never changes, so one static copy can be created and shared for all calls of checkDay

```
// utility function to confirm proper day value based on
// month and year; handles leap years, too
int Date::checkDay( int testDay ) const
{
   static const int daysPerMonth[ monthsPerYear + 1 ] =
      { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

   // determine whether testDay is valid for specified month
   if ( testDay > 0 && testDay <= daysPerMonth[ month ] ){
      return testDay;
   }

   // February 29 check for leap year
   if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
      ( year % 4 == 0 && year % 100 != 0 ) ) ){
      return testDay;
   }

   cout << "Invalid day (" << testDay << ") set to 1.\n";

   return 1; // leave object in consistent state if bad value
}
```

Date member functions

```
// print Date object in form month/day/year
void Date::print() const
{
   cout << month << '/' << day << '/' << year;
}
```

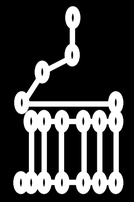
Date.cpp

Back to the Employee class

```
class Employee
{
public:
     Employee( const string &first, const string &last,
               const Date &dateOfBirth, const Date &dateOfHire );
     void print() const;
     ~Employee(); // provided to confirm destruction order
private:
     string firstName;       // composition: member object
     string lastName;        // composition: member object
     const Date birthDate;   // composition: member object
     const Date hireDate;    // composition: member object
};
```

What does the const mean here?

Employee.h



Constant data members

*Use the keyword **const** to specify that an object is not modifiable and that any attempt to modify an object should result in a compilation error.*

But how do we “initialize” constant data members if we can only call constant member functions with the object?

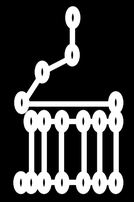
Use a **member initializer list** with the constructor.

Employee class example

```
class Employee
{
public:
    Employee( const string &first, const string &last,
              const Date &dateOfBirth, const Date &dateOfHire );
    void print() const;

private:
    string firstName;        // member object
    string lastName;         // member object
    const Date birthDate;    // constant member object
    const Date hireDate;     // constant member object
};
```

Employee.h



Employee class example

```
Employee::Employee(const string &first, const string &last,
    const Date &dateOfBirth, const Date &dateOfHire)
    : firstName(first),       // initialize firstName
      lastName(last),         // initialize lastName
      birthdate(dateOfBirth), // initialize birthDate
      hireDate(dateOfHire)    // initialize hireDate
{
  // output Employee object to show when constructor is called
  cout << "Employee object constructor: " << firstName
       << ' ' << lastName << endl;
}

Employee::print() const {
    cout << lastName << ", " << firstName << "  Hired: ";
    hireDate.print();
    cout << "  Birthday: ";
    birthDate.print();
}
```

Employee.cpp

birthDate & hireDate are constants
They must be instantiated in the member initializer list.

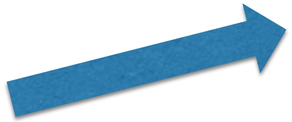
Employee class example

```
int main(){
    Date birth(7, 24, 1949);
    Date hire(3, 12, 1988);
    Employee manager("Bob", "Blue", birth, hire);

    cout << endl;
    manager.print();
    cout << endl;
}
```

main.cpp

Five constructor calls
Two constructor calls were to copy constructor
Nothing was printed in copy constructor



Output

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Blue

Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949
```

Example #2: Cross Country Record Keeping

```
class TrialTimes {
public:
    TrialTimes(string name = "trialRunner", int num = 0) : m_name{name}, m_number{num} { }
    void addTime(double time) { m_times.push_back(time); }
    double getAverageTime( ) const;
    void print() const {cout << m_name << " (# " << m_number << "): "
                                << getAverageTime() << " s" << endl;}
private:
    const string m_name;
    const int m_number;
    vector<double> m_times;
};
```

```
double TrialTimes::getAverageTime( ) const
{
    double total = 0;
    for (const double t : m_times )
    {
        total += t;
    }
    if (m_times.size() > 0)
    {
        total /= m_times.size();
    }
    return total;
}
```

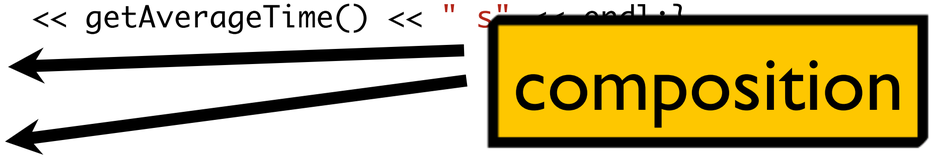
```
int main()
{
    TrialTimes runnerBob("Bob", 1134);
    runnerBob.addTime(20.0);
    runnerBob.addTime(21.6);
    runnerBob.addTime(19.6);
    runnerBob.print();

    TrialTimes runnerSue("Sue", 1532);
    runnerSue.addTime(18.2);
    runnerSue.addTime(17.6);
    runnerSue.addTime(19.5);
    runnerSue.print();

    return 0;
}
```

Example #2

```
class TrialTimes {
public:
    TrialTimes(string name = "trialRunner", int num = 0) : m_name{name}, m_number{num} { }
    void addTime(double time) { m_times.push_back(time); }
    double getAverageTime( ) const;
    void print() const {cout << m_name << " (# " << m_number << "): "
                                << getAverageTime() << " s" << endl;}
private:
    const string m_name;
    const int m_number;
    vector<double> m_times;
};
```



composition

```
double TrialTimes::getAverageTime( ) const
{
    double total = 0;
    for (const double t : m_times )
    {
        total += t;
    }
    if (m_times.size() > 0)
    {
        total /= m_times.size();
    }
    return total;
}
```

```
int main()
{
    TrialTimes runnerBob("Bob", 1134);
    runnerBob.addTime(20.0);
    runnerBob.addTime(21.6);
    runnerBob.addTime(19.6);
    runnerBob.print();

    TrialTimes runnerSue("Sue", 1532);
    runnerSue.addTime(18.2);
    runnerSue.addTime(17.6);
    runnerSue.addTime(19.5);
    runnerSue.print();

    return 0;
}
```

Example #2

```
class TrialTimes {
public:
    TrialTimes(string name = "trialRunner", int num = 0) : m_name{name}, m_number{num} { }
    void addTime(double time) { m_times.push_back(time); }
    double getAverageTime( ) const;
    void print() const {cout << m_name << " (# " << m_number << "): "
                                << getAverageTime() << " s" << endl;}
private:
    const string m_name;
    const int m_number;
    vector<double> m_times;
};
```

constant data members

```
double TrialTimes::getAverageTime( ) const
{
    double total = 0;
    for (const double t : m_times )
    {
        total += t;
    }
    if (m_times.size() > 0)
    {
        total /= m_times.size();
    }
    return total;
}
```

```
int main()
{
    TrialTimes runnerBob("Bob", 1134);
    runnerBob.addTime(20.0);
    runnerBob.addTime(21.6);
    runnerBob.addTime(19.6);
    runnerBob.print();

    TrialTimes runnerSue("Sue", 1532);
    runnerSue.addTime(18.2);
    runnerSue.addTime(17.6);
    runnerSue.addTime(19.5);
    runnerSue.print();

    return 0;
}
```

Example #2

```
class TrialTimes {
public:
    TrialTimes(string name = "trialRunner", int num = 0) : m_name{name}, m_number{num} { }
    void addTime(double time) { m_times.push_back(time); }
    double getAverageTime( ) const;
    void print() const {cout << m_name << " (# " << m_number << "): "
                              << getAverageTime() << " s" << endl;}
private:
    const string m_name;
    const int m_number;
    vector<double> m_times;
};
```

```
double TrialTimes::getAverageTime( ) const
{
    double total = 0;
    for (const double t : m_times )
    {
        total += t;
    }
    if (m_times.size() > 0)
    {
        total /= m_times.size();
    }
    return total;
}
```

constant member functions

```
int main()
{
    TrialTimes runnerBob("Bob", 1134);
    runnerBob.addTime(20.0);
    runnerBob.addTime(21.6);
    runnerBob.addTime(19.6);
    runnerBob.print();

    TrialTimes runnerSue("Sue", 1532);
    runnerSue.addTime(18.2);
    runnerSue.addTime(17.6);
    runnerSue.addTime(19.5);
    runnerSue.print();

    return 0;
}
```

CQ: Consider the Mystery class below. Which of the following member function definitions will compile?

```
class Mystery {
public:
    Mystery(char correctOption)
       : m_correctOption(correctOption)
       { m_other = 'A'; }

    bool isCorrect(char option) const;
    void switchB( ) const;
    void switchC( );

private:
    const char m_correctOption;
    char m_other;
};
```

A.
```
bool Mystery::isCorrect(char option) const
{
   return m_correctOption == option;
}
```

B.
```
void Mystery::switchB( ) const
{
   m_other = m_correctOption;
}
```

C.
```
void Mystery::switchC( )
{
   m_correctOption = m_other;
}
```

D. option A and option B.

E. option A and option C.

F. option B and option C.

Main messages for today

- Composition: you are allowed to use objects generated by other class defintions as data members of a class.
- Constant data members: if you declare a data member as constant, you are not allowed to change its value (after initialization).