DigiPen
INSTITUTE OF TECHNOLOGY

CS170#05.1

Member Functions

Vadim Surov

Outline

- Const Member Functions
- Accessors & Mutators
- Mutable Data Member
- Static Members
- Friends

`const` Member Functions

- You can declare member functions as being const
- Doing so member function promises not to modify member variables when called
- Const member functions can be called on both const and non-const objects
- Can't call non-const member functions on const objects
- Const rule doesn't apply to constructors or destructor
  - Const objects are still created and destroyed. Just can't be modified between creation and destruction

Accessors & Mutators

```
class Date {
public:
  // Accessors
  int get_d() const;
  int get_m() const;
  int get_y() const;
  // Mutators
  void set_d(int);
  void set_m(int);
  void set_y(int);

private:
  int d, m, y;
};
```

- Accessors (“get” methods)
  - Member functions to “read” values
  - Return by value or by const reference
  - Functions often const

Accessors & Mutators (contd)

```
class Date {
public:
  // Accessors
  int get_d() const;
  int get_m() const;
  int get_y() const;
  // Mutators
  void set_d(int);
  void set_m(int);
  void set_y(int);

private:
  int d, m, y;
};
```

- Mutators ("set" methods)
  - Member functions to "write" values
  - Return type is often void
  - Parameter type
    - Same as member variable type
    - const reference to member variable type

Mutable Date Member

- If a data member is declared **mutable**, then it is legal to assign a value to this data member from a const member function

```
class X {
   public:
      bool GetFlag() const {
       accessCount++;  // Legal!
       return flag;
    }
   private:
    bool flag;
    mutable int accessCount;
};
```

Static Members

- Only one copy of the static member is shared by all objects of a class in a program
- Static member is not part of the class object
- Their content is not different from one object of this class to another
- Static members function do not have access to the **this** pointer
- Static integral constants may be initialized within a class declaration by a constant expression
- Static members have the same properties as global variables but they need to indicate the class scope

Static Members Example

```
// static members in classes
class CDummy {
public:
  static int n;
  static const int MAX = 100;
  CDummy () { n++; };
  ~CDummy () { n--; };
};
int CDummy::n=0;
int main () {
  CDummy a;
  CDummy b[5];
  CDummy* c = new CDummy;
  cout << a.n << endl;
  delete c;
  cout << CDummy::n << endl;
  return 0;
}
```

Output: 7 6

Friends

- C++ allows a class to declare its “friends”
  - Offer a limited way to open up class encapsulation
    - Give access to specific classes or functions
- Keyword `friend` is used in class declaration

```
class CRectangle {
    int x, y;
    friend ostream &operator<<(ostream &out,
                const CRectangle &rect);
};
```

Properties Of The Friend Relation

- Friendship gives complete access
  - Friend methods/functions behave like class members
  - public, protected, private scopes are all accessible by friends
- Friendship is asymmetric and voluntary
  - A class gets to say what friends it has (giving permission to them)
  - But one cannot “force friendship” on a class from outside it
- Friendship is not inherited
  - Specific friend relationships must be declared by each class
  - Your parents’ friends are not necessarily your friends