COMSC-200 Lecture Topic 3
Class Members

Reference
Deitel Ch.9.5-9.14

Scope
for non-class functions: file scope
for class members: class scope, too
  direct access in class scope
  need *dot* or *arrow* operators otherwise
using `->` with NULL pointers
  fatal error with data member
  okay with member functions
data member hiding
  scope resolution (`::` or `this->`)
`private` members have class scope

`.cpp` and `.h` files
separating *interface* from *implementation*
possible to distribute compiled `.cpp` with `.h` source

Accessor (Getter) Functions (`const`)
`int getHour() const;` // prototype
  to return a member variable value
  to return a computed value
`bool isEmpty() const;` // prototype
`bool hasUpRiders();` *(Floor class in elevator simulation)*
inline functions: *no inline keyword used*
```
int getHour() const {return hour;}
```

Mutator (Setter) Functions (non-`const`)
used to set data members indirectly
  data protection
  value validation
e.g., `void setHour(int);` // prototype
void functions, or...
  ...return an exit code
  ...return a self reference
inline functions (3 variations):
```
void setHour(int h) {hour = h;}
void setHour(int hour) {this->hour = hour;}
void setHour(int hour) {Time::hour = hour;}
```

Constructor Functions
used to initialize data members
without constructors, data members have *garbage values*
  ...unless brace initialized
the *default* constructor
  declarations: *no parentheses*
constructors with parameter variables
  declarations need parentheses
cannot use brace initialization with constructors
using defaults for parameter variables
  to double as default constructor
  declare in *prototype* (the interface)
inline constructors

When Constructors Are Called
called automatically when object is created
programmer can control *which* constructor is
  called via parameter list

The Destructor Function
use when class members include dynamic memory
  `new` in the constructor requires `delete` in the destructor
e.g., `~Time()`
no parameter variables; no return value
called automatically when object goes out of scope
  ...or is deallocated

Private Functions
utility or *helper* functions
code modules "for internal use only"

Returning References
do *not* return reference to private data member
  violates private-ness
exception: `const` references
use to return *self-reference*
  or ref to an parameter variable

Default Memberwise Assignment
using the = *(assignment)* operator
copies data member values
the problem with pointers & dynamic memory
does not work if there are `const` data members

The Copy Constructor
without copy constructor, does byte-by-byte copy
  but still uses copy constructors
  of member objects that have one
e.g., `Time(const Time&)`
  must be `const` and ampersand
automatically used in pass-by-value, if provided
  instead of memberwise assignment
used when adding to STL container, if provided
good idea to include if objects have...
  ...dynamically-allocated memory
`Time a;` // uses default constructor (if any)
`Time b = a;` // uses copy constructor (if any)
`Time c(a);` // uses copy constructor (if any)

Member Selection
for objects (`Time t;`):
  the *dot operator*: `t.getHour()`
for object pointers (`Time* p;`):
  the *arrow operator*: `p->hour`
can select member functions and data members
  but cannot select constructors or destructors
"host object" comes before the dot or arrow

ABOUT INLINE FUNCTIONS *show / hide*

Using Visual C++ 2010 For Win32 Console Applications