More on C++ Classes

CS3021 Introduction to Data Structures and Intermediate Programming

Class Design (1)

- Make all data members private.
  - This is the default for non-public members.
- Ensure you provide complete set of methods:
  - Constructors: may define several, i.e., overloading.
  - Destructor (only one): compiler creates a default "empty" destructor.
  - Accessor methods: return data member values; also called getters.
  - Mutator methods: modify data members; also called setters.

Class Design (2)

- Other class-specific methods…
  - Predicate methods: answer yes/no questions about data members; usually class specific, e.g., answer "is this flag-style data member set?".
  - Utility functions: useful tools specific to the class data members, such as `print()`.
  - Operator methods (i.e., overloaded): e.g., +, <, <=, >, >=, ==, !=, etc.
  - Finally, any other domain-specific methods that need to be defined for the class.

Private vs. Public Class Members

- By default, data members are private in the class definition, so these two are equivalent:

```
class Time {
  public:
    // public methods...
  private:
    int hour;
};
```

```
class Time {
    int hour;
  public:
    //public methods...
};
```

- Both versions are commonly used, though the explicit version is a good reminder about the "privateness" of data.

Time.cpp

Constructors & Destructor

- Ensure (at least) one default, and one assigned value constructor defined; also, any other domain-specific, e.g., see Ratio class.

```
Time() { hour = 0; }
Time(int h) { hour = h; }
```

- Destructor only really needs to be defined when dynamic memory was allocated, but OK to do explicitly in other cases.

```
~Time() {}
```

Ratio.cpp

Getter/Setter Methods

- Allow only *controlled* access to private data:

```
class Time {
  public:
    void setHour( int h );
    int getHour();
  private:
    int hour;
};
```

```
class Time {
  public:
    int hour;
};
```

This version does not keep the `hour` data "safe".

- Protect the programmer from themself.
  - Ensure any data modifications are constrained to public only methods.

Predicate Methods

- Define predicate methods of class member data:

```
bool isAM ()
{
  if (hour < 12) { return true; }
  else { return false; }
}

bool isPM ()
{
  if (hour >= 12) { return true; }
  else { return false; }
}
```

Domain-specific Methods

- Define methods for other domain-specific functionality:

```
void incrementHour ()
{
  hour++;
}
```

More on Overloading

- Function (method) overloading allowed in C++.
  - We've already seen examples of constructor overloading, which is very common.
- In C++, we can legally define all of these:

```
int   square (int x);
float square (float x);
char* square (char* s, int l);
```

- Multiple functions of the same name must have different *prototypes* (i.e., parameters and return type).

`friend` Functions and Classes (1)

- A `friend` function is defined *outside* the scope of a class (so not a class method), but has access to private members in the class.
- Single functions, or entire classes, may be friends of a class.
- To declare a function as a friend, simply precede it by the keyword **`friend`** within the class.

```
friend void setLocalHour (Time &, int);
```

`friend` Functions and Classes (2)

```
class Time {
  public:
    void setHour(int h);
    int getHour();
  private:
    int hour;
  friend void setLocalHour (Time &, int);
};

void setLocalHour (Time &t, int h)
{
  t.hour = h;
}
```

External `friend` function may access class private member directly.

`friend` Functions and Classes (3)

```
int main ()
{
  Time time1;
  cout << "The hour is " << time1.getHour() << endl;
  time1.setHour(8);
  cout << "The hour is " << time1.getHour() << endl;
  setLocalHour(time1, 12);
  cout << "The hour is " << time1.getHour() << endl;
}
```

- Friendship must be granted; it cannot be taken.
- Friendship is *not* symmetric nor transitive.

Time_friend.cpp

The `this` Pointer (1)

- Keyword `this` identifies a special C++ pointer to an object itself.
- For a class `class` with a declared method `foo()`, when an object of `class` calls the method, `this` refers to the address of that object, within the body of `foo()`.
- You cannot explicitly declare the `this` pointer, since it is a C++ keyword.

```
int Time::getHour()
{
  return this->hour;
}
```

Time_this.cpp

Questions?