C++ for Science and Engineering COSC3000/6000

2018 Spring Semester

Part VIII
Overloading Operators

1 Example : TimeOfDay class

Here we developed the class “TimeOfDay” that manages hours and minutes. Hour is in 24-hours notation.

```
class TimeOfDay
{
public:
    TimeOfDay(int hours, int minutes);
    // precondition : hours and minutes are integers.
    // Initializes the time according to the arguments.
    // call checktime() to convert a right form.

    TimeOfDay(int minutes);
    // precondition : minutes is integers.
    // Initializes the time according to the arguments.
    // call checktime() to convert a right form.

    TimeOfDay();
    // Initialize the time to 00:00

    void output(std::ostream &strm) const;
    // Output time to stream

    int get_hours() const;
    // return hours

    int get_minutes() const;
    // return minutes

    friend bool equal(const TimeOfDay& t1, const TimeOfDay& t2);
    // precondition : t1 and t2 have values.
    // returns true if t1 and t2 represents the same time;
    // otherwise,  return false

    friend TimeOfDay add(const TimeOfDay& t1, const TimeOfDay& t2);
    // precondition : t1 and t2 have values.
    // returns t1 + t2

    friend TimeOfDay subtract(const TimeOfDay& t1, const TimeOfDay& t2);
    // precondition : t1 and t2 have values.
    // returns t1 - t2
private:
    void checktime();
    // precondition : hours and minutes are set
    // Modify hours to be between 0 and 23.
    // Modify minutes to be between 0 and 59.
    // * If minutes<0, add 60 to minutes and subtract 1 from hours,
    //   repeat those until minutes >=0.
    // * If minutes>59, subtract 60 from minutes and add 1 to hours,
    //   repeat those until minutes <= 59.

    // hours
    int hours;
    // minutes
    int minutes;
};
```

In the **TimeOfDay** class, function **add** was used to add two objects of type **TimeOfDay**
In this section we see how to use the '+' operator to make this code legal:

```
        TimeOfDay time1,time2,total;
        :
        total = time + time2;
        // instead of total = add(time,time2);
```

2 Operator Overloading

2.1 Operators As Functions

- An operator is a function used differently than an ordinary function
  - An ordinary function call **enclosed its arguments in parenthesis**
    ```
    add(time,time2)
    ```
  - With a binary operator, the arguments are on **either side of the operator**
    ```
    time + time2
    ```

2.2 Operator Overloading

- Operators can be overloaded
- The definition of operator + for the **TimeOfDay** class is nearly the same as member function **add**
- To overload the + operator for the **TimeOfDay** class
  - Use the name + in place of the name **add**
  - Use keyword **operator** in front of the +
  - Example:
    ```
    friend TimeOfDay operator + (const TimeOfDay& t1...
    ```

2.3 Operator Overloading Rules

- At least one argument of an overloaded operator must be of a class type
- An overloaded operator can be a friend of a class
- New operators cannot be created
- The number of arguments for an operator cannot be changed
- The precedence of an operator cannot be changed
- ., ::, *, and ? cannot be overloaded (multiplication operator * can be overloaded)

2.4 Program Example:Overloading Operators

- equal
  - old version (**sample34.cpp**)

    ```
        friend bool equal(const TimeOfDay& t1, const TimeOfDay& t2);
        // precondition : t1 and t2 have values.
        // returns true if t1 and t2 represents the same time;
        // otherwise,  return false
    ```

  - new version (**sample35.cpp**)

    ```
        friend bool operator ==(const TimeOfDay& t1, const TimeOfDay& t2);
        // precondition : t1 and t2 have values.
        // returns true if t1 and t2 represents the same time;
        // otherwise,  return false
    ```

- add
  - old version (**sample34.cpp**)

    ```
        friend TimeOfDay add(const TimeOfDay& t1, const TimeOfDay& t2);
        // precondition : t1 and t2 have values.
        // returns t1 + t2
    ```

  - new version (**sample35.cpp**)

    ```
        friend TimeOfDay operator + (const TimeOfDay& t1, const TimeOfDay& t2);
        // precondition : t1 and t2 have values.
        // returns t1 + t2
    ```

- subtract
  - old version (**sample34.cpp**)

    ```
        friend TimeOfDay subtract(const TimeOfDay& t1, const TimeOfDay& t2);
        // precondition : t1 and t2 have values.
        // returns t1 - t2
    ```

  - new version (**sample35.cpp**)

    ```
        friend TimeOfDay operator - (const TimeOfDay& t1, const TimeOfDay& t2);
        // precondition : t1 and t2 have values.
        // returns t1 - t2
    ```

2.4.1 Operator Definitions

- equal
  - old version (**sample34.cpp**)

    ```
    bool equal(const TimeOfDay& t1, const TimeOfDay& t2)
    {
        return (t1.hours == t2.hours) && (t1.minutes == t2.minutes);
    }
    ```

  - new version (**sample35.cpp**)

    ```
    bool operator == (const TimeOfDay& t1, const TimeOfDay& t2)
    {
        return (t1.hours == t2.hours) && (t1.minutes == t2.minutes);
    }
    ```

- add

- old version (**sample34.cpp**)

```
TimeOfDay add(const TimeOfDay& t1, const TimeOfDay& t2)
{
    TimeOfDay tsum(t1.hours + t2.hours, t1.minutes + t2.minutes);
    return tsum;
}
```

- new version (**sample35.cpp**)

```
TimeOfDay operator + (const TimeOfDay& t1, const TimeOfDay& t2)
{
    TimeOfDay tsum(t1.hours + t2.hours, t1.minutes + t2.minutes);
    return tsum;
}
```

- subtract
  - old version (**sample34.cpp**)

```
TimeOfDay subtract(const TimeOfDay& t1, const TimeOfDay& t2)
{
    TimeOfDay tsub(t1.hours - t2.hours, t1.minutes - t2.minutes);
    return tsub;
}
```

  - new version (**sample35.cpp**)

```
TimeOfDay operator - (const TimeOfDay& t1, const TimeOfDay& t2)
{
    TimeOfDay tsub(t1.hours - t2.hours, t1.minutes - t2.minutes);
    return tsub;
}
```

2.5 Automatic Type Conversion

- With the right constructors, the system can do **type conversions** for your classes
- This code below actually works

```
        TimeOfDay time1(1,22),time2;
        time2 = time1 + 25;
```

- The integer 25 is converted to type **TimeOfDay** so it can be added to **time1**!
- How does that happen?

2.5.1 Type Conversion Event

- When the compiler sees **time1 + 25**, it first looks for an overloaded + operator to perform **TimeOfDay + integer**
  - If it exists, it might look like this

```
friend TimeOfDay operator + (const TimeOfDay& t1, const int& minutes);
```

- When the appropriate version of + is not found, the compiler looks for a constructor that takes a single integer
  - The **TimeOfDay** constructor that takes a single parameter of type int will work
  - The constructor **TimeOfDay(int minutes)** converts **25** to a **TimeOfDay** object so the two values can be added!
- Although the compiler was able to find a way to add
  time1 + 25
  this addition may not work correctly
  time1 + 25.67

  - There is no constructor in the **TimeOfDay** class that takes a single argument of type **double**

- To permit **time1 + 25.67**, the following constructor should be declared and defined

```
TimeOfDay(double hm);
// initialize hours and minutes
```

2.6 Overloading Unary Operators

- Unary operators take a single argument
- The unary – operator is used to negate a value
  x = -y
- ++ and −− are also unary operators
- Unary operators can be overloaded
  - The **TimeOfDay** class of **sample35.cpp** can includes
    * A binary – operator
    * A unary – operator

2.6.1 A unary – operator for TimeOfDay

- Operator Definition

```
    TimeOfDay operator - ();
    // returns - for both hours and minutes
```

- Operator Implementation

```
TimeOfDay TimeOfDay::operator - ()
{
    TimeOfDay ans(-hours, -minutes);
    return ans;
}
```

2.6.2 overloading ++ and - - unary operators

Remember : difference of **number++ vs ++number**

- (**number**++) returns the current value of **number**, then increments **number**
  - An expression using (**number**++) will use the value of **number** BEFORE it is incremented
- (++**number**) increments **number** first and returns the new value of **number**
  - An expression using (++**number**) will use the value of **number** AFTER it is incremented
- **number** has the same value after either version!

```
int number = 2;
int value_produced = 2 * (number++);
cout << value_produced << " " << number;
```

```
4   3
```

```
int number = 2;
int value_produced = 2 * (++number);
cout << value_produced << " " << number;
```

```
6   3
```

If we define both object++ and ++object operators, we should implement this feature.

- Definitions ++

```
    TimeOfDay operator ++ (); //overloaded prefix ++ operator
    // Incremant 1 minute
    // returns result

    TimeOfDay operator ++ (int); //overloaded postfix ++ operator
    // Increment 1 minute
    // returns the value before increment
```

- Implementations ++

```
TimeOfDay TimeOfDay::operator ++ ()
{
    minutes++;
    checktime();
    TimeOfDay ans(hours,minutes);
    return ans;
}

TimeOfDay TimeOfDay::operator ++ (int)
{
    TimeOfDay ans(hours,minutes);
    minutes++;
    checktime();
    return ans;
}
```