Object-Oriented Programming in C++

August 2012 - Assignment 1

```
class Time
{
    public:
      /* Constructors and destructor */

      /* This is the default constructor
      * This constructor takes no arguments and sets the time to 00:00:00
      * pm
      */

      Time();           // Default constructor

      /* This constructor takes a pre-existing Time object as
      * an argument (parameter) and it is passed by reference to the
      * constructor. It creates a copy of the pre-existing time object
      * which was passed in as an argument.
      */

      Time(Time const& time);         // Copy constructor

      /* This constructor takes as an argument a long integer which
      * represents seconds after midnight. The seconds argument is
      * converted to hours minutes and seconds. The argument is passed
      * by value
      */

      Time(long secondsAfterMidnight);

      /* This constructor takes as an argument a string which is
      * passed to the constructor by value. The argument string is in
      * the format hh:mm:ss. The string should be tested to see if it is
      * in the right format. If it is in the wrong format then the validity
      * flag should be set to false and the hours minutes and seconds
      * data items should be left uninitialized.
```

```
* If the argument string is in the right format, the hours minutes
* and seconds should be checked to see that they have a valid range.
* If the range is invalid e.g. hours greater than 24 or the minutes
* or seconds greater than 59, then the validity flag should be set to
* false and the hours minutes and seconds data items should be left
* uninitialized.
* Note that the description of the boundary conditions in the above
* scenario does not cover all cases where the argument string is
* invalid – can you think of some others?
* If the hours minutes and seconds from the argument string are
* valid, the valid flag should be set to true and the hours minutes
* and seconds should be initialised to the passed in values. If you
* are storing the time as seconds after midnight, then that value
* should be calculated and stored.
*/

Time(char const* tstring);   // String in hh:mm:ss
                             // format (24 hour time).


~Time();                     // Destructor

/* Const (read-only) functions
* The following functions are all “getter” functions that return a
* value.

* The GetTime function returns a string representation of the time
* The Boolean argument is optional. If a Boolean value is passed in
* as an argument to the function and its value is true then the
* string returned by the function is in 24 hour format.
* If military then hh:mm:ss else hh:mm:ss am.

char* GetTime(bool military = false) const;

/* The GetHour function returns the hour value in a Time object as
* an integer. There is no argument to the function.
*/
int   GetHour() const;        // Get hour value.

/* The GetMinute function returns the minute value in a Time object
```

```
* as an int. There is no argument to the function.
*/

int   GetMinute() const;      // Get minute value.

/* The GetSecond function returns the hour value in a Time object as
* an int. There is no argument to the function.
*/

int   GetSecond() const;      // Get second value.

/* The operator ! is used to determine if a time object
*  is valid. It returns the value of the valid field.
*/

bool  operator !() const;     // Returns true if time
                              //     is NOT valid

/* The IsAM() function takes no arguments and returns a Boolean
* value of true if the time object it refers to has an AM time
* If the time is PM or the time object is invalid the value of false
* is returned.
*/
bool  IsAM() const;           // Is time AM?

/* The == operator returns a value of true if all the stored
* attributes of the two objects are equal.
*/
bool  operator ==(Time const& time) const; // Are times equal?

/* The + operator returns a time object. The time stored in the
* returned object is the sum of the hours, the minutes and the
* seconds. Note the following :
* (1) The addition of the seconds may result in a seconds value
* greater than 60. In this case there is a carry over of 1 to the
* minutes addition.
* (2) Similarly to point (1) the addition of the minutes may result
* in a minutes value greater than 60. In this case there is a carry over
* of 1 to the hours addition.
* (3) The addition of the hours may result in a sum of hours greater
```

```
* than 24. This is an error condition known as an overflow.
*
* If either of the time objects being added together has a valid
* flag of false then the returned object is false and its hours
* minutes and seconds values are not initialised. If the returned object
* stores the seconds since midnight value it is not initialised and the
* valid flag is set to false..
*
* If there is an overflow in the result of the addition then the valid
* flag of the object returned is set to false and all other attributes
* of the returned object are uninitialized.
*/
Time  operator +(Time const& time) const;  // Add times.

/* The - operator returns a time object. The time stored in the
* returned object is the difference of the value of the hours, the
* minutes and the seconds in the first object minus the hours,
* minutes and seconds in the second object. Note the following :
* (1) The subtraction of the seconds may result in a seconds value
* less than 0. In this case there is a borrow of 1 from the
* minutes in the first time object.
* (2) Similarly to point (1) the subtraction of the minutes may result
* in a minutes value less than 0. In this case there is a borrow of 1
* from the hours column.
* (3) The subtraction of the hours may result in a value of hours
* less than 0. This is an error condition known as an underflow.
*
*
* If either of the time objects being subtracted has a valid
* flag of false then the returned object is false and its hours
* minutes and seconds values are not initialised. If the returned object
* stores the seconds since midnight value it is not initialised.
*
* If there is an underflow in the result of the subtraction then the
* valid flag of the object returned is set to false and all other
* attributes of the returned object are uninitialized.
*/
Time  operator -(Time const& time) const;  // Subtract times.

/* Non-const (read/write) functions
```

```
* The SetTime function returns no value. It takes 3 integer
* arguments, the hrs, mins and secs. If the function is called with
* only one argument it corresponds to the hrs argument, and the mins
* and secs
* arguments are set to their default values of 0. In the event of
* invalid values being passed in as arguments, the valid flag for the
* object is set to false and no changes will occur to the object's
* other data items.
*/

void  SetTime(int hrs, int mins = 0, int secs = 0);

/* The AddHours function takes one integer argument. The integer can have
* a range between -24 and +24. If the argument is outside this range it
* should set the valid flag of the object to false and not change the value
* of the hours data item.
* If the argument is positive and within the acceptable range, the hours
* argument is added to the existing hours data item in the object. If the
* argument is negative and within the acceptable range, it is subtracted
* from the existing hours argument.
* It is possible for overflow or underflow to occur where the resulting
* time is outside the range 00:00:00 to 24:00:00. In this case no change
* should occur in the hours, minutes or seconds data item and the valid
* flag should be set to false.
*
*/
void  AddHours(int hours);          // Add hours (which may be <0).

/* The AddMinutes function takes one integer argument. The integer can have
* a range between -1440 and +1440. There are 1440 minutes in a day. If the
* value passed in is outside this range, then underflow or overflow will
* occur. In this case you should set the valid flag of the object to false
* and not change the value of the minutes data item.
*
* If the argument is positive and within the acceptable range, the minutes
* argument is added to the existing hours data item in the object. If the
* argument is negative and within the acceptable range, it is subtracted
* from the existing hours argument.
* It is possible for overflow or underflow to occur in the addition or
```

```
      * subtraction operation. In many cases this will be acceptable and the
      * hours or seconds data item should also be altered to allow for the
      * overflow or underflow.
      * In the remaining cases of overflow or underflow, where the resulting time
      * is outside the range 00:00:00 to 24:00:00 no change should occur
      * in the hours, minutes or seconds data item and the valid flag should be
      * set to false.
      */

      void  AddMinutes(int minutes);      // Add minutes (which may be < 0).

      /* The AddSeconds function takes one integer argument. The integer can have
      * a range between -86400 and +86400. There are 86400 seconds in a day. If
      * the value passed in is outside this range, then underflow or overflow
      * will occur. In this case you should set the valid flag of the object to
      * false and not change the value of the minutes data item.
      *
      * If the argument is positive and within the acceptable range, the minutes
      * argument is added to the existing hours data item in the object. If the
      * argument is negative and within the acceptable range, it is subtracted
      * from the existing hours argument.
      * It is possible for overflow or underflow to occur in the addition or
      * subtraction operation. In many cases this will be acceptable and the
      * hours or seconds data item should also be altered to allow for the
      * overflow or underflow.
      * In the remaining cases of overflow or underflow, where the resulting time
      * is outside the range 00:00:00 to 24:00:00 no change should occur
      * in the hours, minutes or seconds data item and the valid flag should be
      * set to false.
      */

      void  AddSeconds(int seconds);      // Add seconds (which may be < 0).

   Private:
      /* the data items storing the hours, minutes and seconds or the seconds
      * since midnight go here. Any private functions you write will also go
      * in this section.
      */
};
```