

```

//
//  main.cpp
//  AsboloteCpp_ch11_1
//

//This is the application file: timedemo.cpp which demonstrates use of
  DigitalTime.
//This is the same file as 11-03.cpp

#include <iostream>
#include "dtime.h"

void readHour(int& theHour);

int main( )
{
    using std::cout;
    using std::cin;
    using std::endl;
    using DTimeSavitch::DigitalTime;

    int theHour;

    readHour(theHour);

    DigitalTime clock(theHour, 0), oldClock;

    oldClock = clock;
    clock.advance(15);
    if (clock == oldClock)
        cout << "Something is wrong.";
    cout << "You entered " << oldClock << endl;
    cout << "15 minutes later the time will be "
    << clock << endl;
    clock.advance(2, 15);
    cout << "2 hours and 15 minutes after that\n"
    << "the time will be "
    << clock << endl;
    return 0;
}

void readHour(int& theHour)
{
    using std::cout;
    using std::cin;

    cout<< "Let's play a time game.\n"
    << "Let's pretend the hour has just changed.\n"
    << "You may write midnight as either 0 or 24,\n"
    << "but, I will always write it as 0.\n"
    << "Enter the hour as a number (0 to 24): ";
    cin >> theHour;
}

```

```

//This is the header file dtime.h. This is the interface for the class
DigitalTime.
//Values of this type are times of day. The values are input and output in
//24 hour notation as in 9:30 for 9:30 AM and 14:45 for 2:45 PM.

//This is the same file as 11-01.cpp

#ifndef DTIME_H
#define DTIME_H

#include <iostream>
using namespace std;

namespace DTimeSavitch
{
    class DigitalTime
    {
    public:
        DigitalTime(int theHour, int theMinute);
        DigitalTime( );
        //Initializes the time value to 0:00 (which is midnight).

        int getHour( ) const;
        int getMinute( ) const;
        void advance(int minutesAdded);
        //Changes the time to minutesAdded minutes later.

        void advance(int hoursAdded, int minutesAdded);
        //Changes the time to hoursAdded hours plus minutesAdded minutes
        later.

        friend bool operator ==(const DigitalTime& time1,
                                const DigitalTime& time2);

        friend istream& operator >>(istream& ins, DigitalTime& theObject);

        friend ostream& operator <<(ostream& outs, const DigitalTime&
            theObject);

    private:

        int hour;
        int minute;

        static void readHour(int& theHour);
        //Precondition: Next input in to be read from the keyboard is
        //a time in notation, like 9:45 or 14:45.
        //Postcondition: theHour has been set to the hour part of the time.
        //The colon has been discarded and the next input to be read is the
        minute.

        static void readMinute(int& theMinute);

```

```
//Reads the minute from the keyboard after readHour has read the hour.
```

```
static int digitToInt(char c);
```

```
//Precondition: c is one of the digits 00 through 99.
```

```
//Returns the integer for the digit; that is, digitToInt(03) returns 3.
```

```
};
```

```
}//DTimeSavitch
```

```
#endif //DTIME_H
```

```
//This is the implementation file: dttime.cpp of the class DigitalTime.  
//The interface for the class DigitalTime is in the header file dttime.h.  
// This is the same as 11-02.cpp
```

```
#include <iostream>  
#include <cctype>  
#include <cstdlib>  
using namespace std;  
#include "dttime.h"  
  
namespace{  
  
    int digitToInt(char c)  
    {  
        return ( int(c) - int('0') );  
    }  
  
    //Uses iostream, cctype, and cstdlib:  
    void readMinute(int& theMinute)  
    {  
        char c1, c2;  
        cin >> c1 >> c2;  
  
        if (!(isdigit(c1) && isdigit(c2)))  
        {  
            cout << "Error illegal input to readMinute\n";  
            exit(1);  
        }  
  
        theMinute = digitToInt(c1)*10 + digitToInt(c2);  
  
        if (theMinute < 0 || theMinute > 59)  
        {  
            cout << "Error illegal input to readMinute\n";  
            exit(1);  
        }  
    }  
  
    //Uses iostream, cctype, and cstdlib:  
    void readHour(int& theHour)  
    {  
        char c1, c2;  
        cin >> c1 >> c2;  
        if ( !( isdigit(c1) && (isdigit(c2) || c2 == ':' ) ) )  
        {  
            cout << "Error illegal input to readHour\n";  
            exit(1);  
        }  
  
        if (isdigit(c1) && c2 == ':')  
        {  
            theHour = digitToInt(c1);  
        }  
        else //(isdigit(c1) && isdigit(c2))  
        {
```

```

        theHour = digitToInt(c1)*10
        + digitToInt(c2);
        cin >> c2; //discard ':'
        if (c2 != ':')
        {
            cout << "Error illegal input to readHour\n";
            exit(1);
        }
    }

    if (theHour == 24)
        theHour = 0; //Standardize midnight as 0:00

    if ( theHour < 0 || theHour > 23 )
    {
        cout << "Error illegal input to readHour\n";
        exit(1);
    }
}

} //unnamed namespace

namespace DTimeSavitch
{
    //Uses iostream and cstdlib:
    DigitalTime::DigitalTime(int theHour, int theMinute)
    {
        if (theHour < 0 || theHour > 24 || theMinute < 0 || theMinute > 59)
        {
            cout << "Illegal argument to DigitalTime constructor.";
            exit(1);
        }
        else
        {
            hour = theHour;
            minute = theMinute;
        }

        if (hour == 24)
            hour = 0; //standardize midnight as 0:00
    }

    DigitalTime::DigitalTime( )
    {
        hour = 0;
        minute = 0;
    }

    int DigitalTime::getHour( ) const
    {
        return hour;
    }

    int DigitalTime::getMinute( ) const

```

```

{
    return minute;
}

void DigitalTime::advance(int minutesAdded)
{
    int grossMinutes = minute + minutesAdded;
    minute = grossMinutes%60;

    int hourAdjustment = grossMinutes/60;
    hour = (hour + hourAdjustment)%24;
}

void DigitalTime::advance(int hoursAdded, int minutesAdded)
{
    hour = (hour + hoursAdded)%24;
    advance(minutesAdded);
}

bool operator ==(const DigitalTime& time1, const DigitalTime& time2)
{
    return (time1.hour == time2.hour && time1.minute == time2.minute);
}

//Uses ostream:
ostream& operator <<(ostream& outs, const DigitalTime& theObject)
{
    outs << theObject.hour << ':';
    if (theObject.minute < 10)
        outs << '0';
    outs << theObject.minute;
    return outs;
}

//Uses istream:
istream& operator >>(istream& ins, DigitalTime& theObject)
{
    readHour(theObject.hour);
    readMinute(theObject.minute);
    return ins;
}

int DigitalTime::digitToInt(char c)
{
    return ( int(c) - int('0') );
}

} //DTimeSavitch

```