CSCI 2170 Abstract Data Type, C++ class

- In many cases, the solution to a problem requires operation on data, for example add, sort, print, and remove operations on data. In these cases, it is better to put the focus on data, and think in terms of what we can do with a collection of data, INDEPENDENTLY of HOW we do it. This is called **data abstraction**. Data abstraction is the fundamental idea of Object Oriented Programming.

- **Abstract Data Type (ADT)** – a collection of data together with a set of operations on that data
  ADT is different from Data Structure – data structure is a construct within a programming language that stores a collection of data. It defines how data is stored in the memory for a program.

- When constructing an ADT, there are two steps involved:
  - Specification – indicate precisely what each operation of an ADT does.
  - Implementation –specify how each operation is implemented. This includes selecting a data structure for data involved in ADT using programming language → data structure is part of an ADT's implementation

As a client, using an ADT is like using a vending machine:
we do not grab the snacks directly, we do not care how the snacks are stored..,
we press a button, and expect the corresponding snack to come out.
➔
we request for an operation of an ADT, the operation outputs the appropriate result.
We do not need to know how the result is generated, nor how data is stored in the ADT.

- **ADT implemented as class:**

class, object
- class is the ADT implemented in C++
- object is an instance of a class

public, private, const
- Data and methods declared in the public section can be accessed by the client program of the class
- Data and methods declared in the private section can not be accessed by the client program of the class
- A const method may not modify the data of a class

constructor (default constructor, other constructors)
- activated when an object of the class is created
- used to initialize data of the object
- a class can have more than one constructor

destructor
- activated when an object of the class exits its scope
- a class can only have one destructor
- destructor is ONLY necessary in a class when data of the class has acquired dynamically allocated memory. In this case, destructor is

methods -- member function
accessors – get methods
mutators – set methods

- Use **#ifndef / #define / #endif** to prevent multiple inclusion of a class definition

ADT Time Example **Header file**

```
#ifndef   TIMETYPE_H
#define   TIMETYPE_H
class TimeType
{
public:
   // default constructor
   // Postcondition:  Class object is constructed  &&  Time is 0:0:0
   TimeType();

   // value constructor
   // Precondition:      0 <= initHrs <= 23  &&  0 <= initMins <= 59  && 0 <= initSecs <= 59
   // Postcondition:      Class object is constructed. Time is set according to the incoming parameters
   TimeType( int initHrs,  int initMins, int initSecs );

    // Precondition:         0 <= hours <= 23  &&  0 <= minutes <= 59  && 0 <= seconds <= 59
    // Postcondition:      Time is set according to the incoming parameters
   void SetTime( int hours, int minutes, int seconds );

   // Postcondition: hour value is set to hours
   void SetHour(int hours);

   // Postconsition: the hour value is returned back
   int GetHour()  const;

   // Postcondition:  Time has been advanced by one second, with  23:59:59 wrapping around to 0:0:0
   void Increment();

   // Postcondition:    Time has been output in the form HH:MM:SS
   void Write() const;

  // Postcondition:      Function value == true, if this time equals otherTime;    == false, otherwise
  bool Equal( TimeType  otherTime ) const;

   // Precondition:  This time and otherTime represent times in the same day
   // Postcondition:   Function value == true,  if this time is earlier in the day than otherTime;
   // == false, otherwise
  bool LessThan(TimeType otherTime ) const;

private:
    int hrs;
    int mins;
    int secs;
};
#endif
```

Implementation file

```
#include "timetype.h"
#include <iostream>
using namespace std;

TimeType::TimeType()
{
   hrs = 0;
```

```
    mins = 0;
    secs = 0;
}

TimeType::TimeType( int initHrs,   int initMins, int initSecs ) {
    hrs = initHrs;
    mins = initMins;
    secs = initSecs;
}

void TimeType::SetTime( int hours, int minutes,  int seconds ) {
    hrs = hours;
    mins = minutes;
    secs = seconds;
}

void TimeType::SetHour(int hours) {
    hrs = hours;
}

int TimeType::GetHour() const{
   return hrs;
}

void TimeType::Increment()  {
    secs++;
    if (secs > 59)     {
        secs = 0;
        mins++;
        if (mins > 59)  {
            mins = 0;
            hrs++;
            if (hrs > 23)
                hrs = 0;
        }
    }
}

void TimeType::Write() const   {
    if (hrs < 10)
        cout << '0';
    cout << hrs << ':';
    if (mins < 10)
        cout << '0';
    cout << mins << ':';
    if (secs < 10)
        cout << '0';
    cout << secs;
}

bool TimeType::Equal(TimeType otherTime ) const    {
    return (hrs == otherTime.hrs && mins == otherTime.mins && secs == otherTime.secs);
}
```

```
bool TimeType::LessThan(TimeType otherTime ) const    {
   return (hrs < otherTime.hrs ||  hrs == otherTime.hrs && mins < otherTime.mins ||
           hrs == otherTime.hrs && mins == otherTime.mins && secs < otherTime.secs);
}
```

Client Program

```
#include "timetype.h"
#include <iostream>
using namespace std;
int main()
{
   TimeType time1(5, 30, 0);
   TimeType time2;
   int      count;

   cout << "time1: ";
   time1.Write();

   time2.SetTime(4, 20, 3);
   cout << "  time2: ";
   time2.Write();
   cout << endl;
   cout << "  time2 hour is: "<< time2.GetHour() << endl;

   time2.SetHour(5);
   time2.Increment();
   cout << "New time2: ";
   time2.Write();
   cout << endl;

   if (time1.Equal(time2))
      cout << "Times are equal" << endl;
   else
      cout << "Times are NOT equal" << endl;

   if (time1.LessThan(time2))
      cout << "time1 is less than time2" << endl;
   else
      cout << "time1 is NOT less than time2" << endl;

   if (time2.LessThan(time1))
      cout << "time2 is less than time1" << endl;
   else
      cout << "time2 is NOT less than time1" << endl;

   cout << "Incrementing time1:" << endl;
   time1.SetTime(23, 59, 55);
   for (count = 1; count <= 10; count++)     {
      time1.Write();
      cout << endl;
      time1.Increment();
   }
   return 0;
}
```