```
/*
  File: temps.cpp
  Created by: Tan Qi Hao
  Creation Date: 4/20/2019
  Synopsis: This program collect a list of temperatures samples from the
users
  throughout a single dat and write the result into a files.
*/

#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>

using namespace std;

class MilTime
{
private:
      int hour;
      int minutes;

public:
      int getHour() const;
      int getMin() const;
      void setHour(const int h);
      void setMin(const int m);

      void write_out(ofstream & fout);
};

class Fahrenheit
{
private:
      double degree;
      MilTime time;

public:
      // member functions
      double getTemp() const;
      MilTime getTime() const;
      double getCelsius() const;

      void setTemp(const double temp);
      void setTime(const int h, const int m);
};

// FUNCTION PROTOTYPES GO HERE:
string read_filename(const string prompt);
int read_num_samples(const string prompt);
Fahrenheit read_sample();
```

```
void write_to_file(const string filename, const vector<Fahrenheit> &
samples);
double average_temp(const vector<Fahrenheit> samples);
double coldest_temp(const vector<Fahrenheit> samples);
MilTime last_sample(const vector<Fahrenheit> samples);

int main()
{

      // Define local variables
      string fname;          // Output file name
      int n;                       // Number of temperature samples
      vector<Fahrenheit> temps;    // Temperature samples

      // Prompt for the name of the output file to write
      fname = read_filename("Enter the output file name: ");

      // Prompt for the number of temperature samples
      n = read_num_samples("Enter the number of samples: ");

      // Read temperature samples from user
      for (int i = 0; i < n; i++) {
                  cout << endl;
                  temps.push_back(read_sample());
      }

      // Write the sample information to the outputfile
      write_to_file(fname, temps);
      cout << endl;

      return 0;
}

// FUNCTION DEFINITIONS GO HERE:

// CLASS MEMBER FUNCTION DEFINITINOS GO HERE:

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

int MilTime::getMin() const
{
  return minutes;
}

void MilTime::setHour(const int h)
{
  hour = h;
}

void MilTime::setMin(const int m)
{
```

```
  minutes = m;
}

void MilTime::write_out(ofstream & fout)
{

  if(getHour() >= 0 && getHour() < 10){

      fout << "0" << getHour();

    }

    else{

      fout << getHour();

    }

    fout << ":";


if(getMin() >= 0 && getMin() < 10){

      fout << "0" << getMin();

    }

    else{

      fout << getMin();

    }

}

double Fahrenheit::getTemp() const
{
  return degree;
}

MilTime Fahrenheit::getTime() const
{
  return time;
}

double Fahrenheit::getCelsius() const
{

return (degree - 32) * 5/9;

}

void Fahrenheit::setTemp(const double temp)
{
```

```
  degree = temp;
}

void Fahrenheit::setTime(const int h, const int m)
{
  time.setHour(h);
  time.setMin(m);
}

string read_filename(const string prompt)
{

  string file_name; //input file name

  cout << prompt;
  cin >> file_name;

  return file_name;

}

int read_num_samples(const string prompt)
{
  int num_samples; //input number of samples

  cout << prompt;
  cin >> num_samples;

 return num_samples;
}

Fahrenheit read_sample()
{
  Fahrenheit temp_time; //Temperature and time in fahrenheit class
  double tp; //input temperature
  int hour, min; //input hour and minutes

  cout << "Enter degrees(Fahrenheit): ";
  cin >> tp;

 cout << "Enter hours (Military time):";
 cin >> hour;

 cout << "Enter minutes (Military time):";
 cin >> min;

 temp_time.setTime(hour, min);
 temp_time.setTemp(tp);

  return temp_time;
}

void write_to_file(const string filename, const vector<Fahrenheit> &
samples)
```

```
{
  ofstream fout;

  fout.open(filename.c_str(), ios::out);

  if(!fout.is_open()){

    cerr << "Unable to open file " << filename << endl;
    exit(10);

  }

  fout << endl;
  fout << "-------------------------------------------------------------" <<
endl;

  for(int i = 0; i < samples.size(); i++){

    fout << "Sample #" << i + 1 << ": " << samples[i].getTemp() << "
degrees F (or "
       << samples[i].getCelsius() << " degrees C ) at ";
       samples[i].getTime().write_out(fout);
       fout << endl;
  }

 fout << "-------------------------------------------------------------" <<
endl;

 fout << "The average temperature is "<< average_temp(samples) << "
degrees F" << endl;
 fout << "The coldest temperature is " << coldest_temp(samples) << "
degrees F" << endl;
 fout << "The last sample was taken at time ";
 last_sample(samples).write_out(fout);

  fout.close();

}

double average_temp(const vector<Fahrenheit> samples)
{
  double sum(0.0); //Sum of all the number
  double average; // average of the sum

  for(int i = 0; i < samples.size(); i++){

    sum = sum + samples[i].getTemp();

  }

  average = sum / samples.size();

  return average;
```

```
}

double coldest_temp(const vector<Fahrenheit> samples)
{
  double lowtemp(1000000); //lowest temperature

  for(int i = 0; i < samples.size(); i++){

    if(samples[i].getTemp() < lowtemp){

      lowtemp = samples[i].getTemp();

  }

 }

  return lowtemp;

}

MilTime last_sample(const vector<Fahrenheit> samples)
{
  MilTime last_time; //the last sample time

  last_time.setHour(0);
  last_time.setMin(0);

   for(int i = 0; i < samples.size(); i++){

     if(samples[i].getTime().getHour() >= last_time.getHour() ||
      (samples[i].getTime().getHour() == last_time.getHour()
       && samples[i].getTime().getMin() > last_time.getMin())){

       last_time = samples[i].getTime();

     }

   }

   return last_time;

}
```