Welcome to C++

CSE 225 - Data Structures and Algorithms

Md. Mahfuzur Rahman
ECE Department
North South University

1 frac.h

```
class FractionType
{
public:
  void Initialize(int numerator, int denominator);
  // Function: Initialize the fraction
  // Pre:  None
  // Post: Fraction is initialized
  int NumeratorIs();
  // Function: Returns the value of the numerator
  // Pre:  Fraction has been initialized
  // Post: numerator is returned
  int DenominatorIs();
  // Function: Returns the value of the denominator
  // Pre:  Reaction has been initialized
  // Post: denominator is returned
  bool IsZero();
  // Function: Determines if fraction is zero
  // Pre:  Fraction has been initialized
  // Post: Returns true if numerator is zero
  bool IsNotProper();
  // Function: Determines if fraction is a proper fraction
  // Pre:  Fraction has been initialized
  // Post: Returns true if fraction is greater than one
  int ConvertToProper();
  // Function: Converts the fraction to a whole number and a
  //       fractional part
  // Pre:  Fraction has been initialized, is in reduced form, and
  //       is not a proper fraction
  // Post: Returns whole number
  //       Remaining fraction is original fraction minus the
  //       whole number; fraction is in reduced form
private:
  int num;
  int denom;
};
```

2 frac.cpp

```
// Implementation file for class FractionType
#include "frac.h"
void FractionType::Initialize(int numerator, int denominator)
// Function: Initialize the fraction
// Pre:  None
// Post: numerator is stored in num; denominator is stored in
//       denom
{
  num = numerator;
  denom = denominator;
}
int FractionType::NumeratorIs()
// Function: Returns the value of the numerator
{
  return num;
}
int FractionType::DenominatorIs()
// Function: Returns the value of the denominator
{
  return denom;
}

bool FractionType::IsZero()
// Function: Determines if fraction is zero
// Pre:  Fraction has been initialized
// Post: Returns true if numerator is zero
{
  return (num == 0);
}

bool FractionType::IsNotProper()
// Function: Determines if fraction is a proper fraction
// Pre:  Fraction has been initialized
// Post: Returns true if num is greater than or equal to denom
{
  return (num >= denom);
}

int FractionType::ConvertToProper()
// Function: Converts the fraction to a whole number and a
//        fractional part
// Pre:  Fraction has been initialized , is in reduced form, and
//        is not a proper fraction
// Post: Returns num divided by denom
//       num is original num % denom; denom is not changed
{
  int result;
  result = num / denom;
  num = num % denom;
  return result;
}
```

3 fracDr.cpp

```cpp
// Test driver
#include <iostream>
#include <fstream>
#include <string>
#include "frac.h"
int main()
{
  using namespace std;
  ifstream inFile;        // file containing operations
  ofstream outFile;       // file containing output
  string inFileName;      // input file external name
  string outFileName;     // output file external name
  string outputLabel;
  string command;         // operation to be executed
  int numCommands;
  FractionType fraction;
  // Prompt for file names, read file names, and prepare files
  cout << "Enter name of input file; press return." << endl;
  cin  >> inFileName;
  inFile.open(inFileName.c_str());

  cout << "Enter name of output file; press return." << endl;
  cin  >> outFileName;
  outFile.open(outFileName.c_str());

  cout << "Enter name of test run; press return." << endl;
  cin  >> outputLabel;
  outFile << outputLabel << endl;

  inFile >> command;
  numCommands = 0;
  while (command != "Quit")
  {
    if (command == "Initialize")
    {
      int numerator, denominator;
      inFile  >> numerator;
      inFile  >> denominator;
      fraction.Initialize(numerator, denominator);
      outFile <<  "Numerator: "  << fraction.NumeratorIs()
              <<  " Denominator: " << fraction.DenominatorIs() << endl;
    }
    else if (command == "NumeratorIs")
      outFile <<  "Numerator: "  << fraction.NumeratorIs() << endl;
    else if (command == "DenominatorIs")
      outFile << "Denominator: " << fraction.DenominatorIs() << endl;
    else if (command == "IsZero")
      if (fraction.IsZero())
        outFile << "Fraction is zero " << endl;
      else
        outFile << "Fraction is not zero " << endl;
    else if (command == "IsNotProper")
      if (fraction.IsNotProper())
        outFile << "Fraction is improper " << endl;
      else
        outFile << "Fraction is proper " << endl;
    else
    {
      outFile << "Whole number is " << fraction.ConvertToProper()
              << endl;
      outFile <<  "Numerator: "  << fraction.NumeratorIs()
              <<   " Denominator: " << fraction.DenominatorIs() << endl;

    }

    numCommands++;
    cout << "Command number " << numCommands << " completed."
         << endl;
    inFile >> command;
  };

  cout << "Testing completed."  << endl;
  return 0;
}
```

4 fracIn

```
Initialize
3
4
IsZero
IsNotProper
NumeratorIs
DenominatorIs
Initialize
4
3
IsNotProper
ConvertToProper
Initialize
0
1
IsZero
Initialize
8
4
IsNotProper
ConvertToProper
Quit
```

5 fracOut

```
FinalRun
Numerator: 3 Denominator: 4
Fraction is not zero
Fraction is proper
Numerator: 3
Denominator: 4
Numerator: 4 Denominator: 3
Fraction is improper
Whole number is 1
Numerator: 1 Denominator: 3
Numerator: 0 Denominator: 1
Fraction is zero
Numerator: 8 Denominator: 4
Fraction is improper
Whole number is 2
Numerator: 0 Denominator: 4
```

6 DateType.h

```cpp
#include <string>
#include <fstream>
using namespace std;
// Declare a class to represent the Date ADT
// This is file DateType.h.
enum RelationType {LESS, EQUAL, GREATER};
// Compares self with someDate.
class DateType
{
public:
  void Initialize(int newMonth, int newDay, int newYear);
  int GetMonth() const;              // returns year
  int GetYear() const;               // returns month
  int GetDay() const;              // returns day
  string GetMonthAsString() const;          // returns month as a string
  DateType Adjust(int daysAway) const;
  RelationType ComparedTo(DateType someDate) const;
private:
  int  year;
  int  month;
  int  day;
};
```

7 DateType.cpp

```cpp
// File DateType.cpp  contains the implementation of class DateType
#include "DateType.h"
#include <fstream>
#include <iostream>
using namespace std;

// Nmber of days in each month
static int daysInMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30,
                            31, 30, 31};

// Nmaes of the months
static string conversionTable[] = {"Error", "January", "February",
    "March", "April", "May", "June", "July", "August", "September",
    "October", "November", "December"};

void DateType::Initialize(int newMonth, int newDay, int newYear)
// Post: If newMonth, newDay and newYear represent a valid date,
//       year is set to newYear;
//       month is set to newMonth;
//       day is set to newDay;
//       otherwise a string exception is thrown, stating the
//       first incorrect parameter.
{
  if (newMonth < 1 || newMonth > 12)
     throw string("Month is invalid");
  else if (newDay < 1 || newDay > daysInMonth[newMonth])
     throw string("Day is invalid");
  else if (newYear < 1583)
     throw string("Year is invalid");
  year = newYear;
  month = newMonth;
  day = newDay;
}
int DateType::GetMonth() const
// Accessor function for data member month.
{
   return month;
}

string DateType::GetMonthAsString() const
// Returns data member as a string
{
  return conversionTable[month];
}

int DateType::GetYear() const
// Accessor function for data member year.
{
  return year;
}

```

```
int DateType::GetDay() const  // Accessor function for data member day.
{
  return day;
}

RelationType DateType::ComparedTo(DateType aDate) const
// Post: Function value = LESS, if self comes before aDate.
//                      = EQUAL, if self is the same as aDate.
//                      = GREATER, if self comes after aDate.
{
  if (year < aDate.year)
    return LESS;
  else if (year > aDate.year)
    return GREATER;
  else if (month < aDate.month)
    return LESS;
  else if (month > aDate.month)
    return GREATER;
  else if (day < aDate.day)
    return LESS;
  else if (day > aDate.day)
    return  GREATER;
  else return EQUAL;
}

DateType DateType::Adjust(int daysAway) const
// Post: Function value = newDate daysAway from self
{
  int newDay = day + daysAway;
  int newMonth = month;
  int newYear = year;
  bool finished = false;
  int daysInThisMonth;
  DateType returnDate;
  while (! finished)
  {
    daysInThisMonth = daysInMonth[newMonth];
   if (newMonth == 2)
     if (((newYear % 4 == 0) && !(newYear % 100 == 0))
          || (newYear % 400 == 0))
       daysInThisMonth++;
   if (newDay <= daysInThisMonth)
     finished = true;
   else
   {
     newDay = newDay - daysInThisMonth;
     newMonth = (newMonth % 12) + 1;
     if (newMonth == 1)
       newYear++;
   }
  }
  returnDate.Initialize(newMonth, newDay, newYear);
  return returnDate;
}
```

8 DateDr.cpp

```cpp
/*
 *  Main.cpp
 *  DateType
 */

#include <fstream>
#include <string>
#include <iostream>
#include "DateType.h"
using namespace std;
int main()
{
  string command;
  int month, day, year;
  DateType date, date2;
  ofstream outFile;
  int daysAway;

  outFile.open("date.out");
  cout << "Input a command or Quit to terminate the test" << endl;
  cin >> command;

  while (command != "Quit")
  {
    if (command == "Initialize")
  {
    cout << "Input a month, day, and year on one line" << endl;
    cin >> month >> day >> year;
    try
    {
      date.Initialize(month, day, year);
    outFile << command << ": " << date.GetMonthAsString() << " " << date.
   GetDay() << ", "<< date.GetYear() << endl;
      }
    catch(string msg)
    {
    outFile << msg << endl;
      }
  }
  else if (command == "GetMonth")
    outFile << command << ": "<< date.GetMonth() << endl;
  else if (command == "GetMonthAsString")
    outFile << command << ": " << date.GetMonthAsString() << endl;
  else if (command == "GetDay")
    outFile << command << ": " << date.GetDay();
  else if (command == "GetYear")
    outFile << command << ": "<< date.GetYear();
  else if (command == "ComparedTo")
    {
    cout << "Input a month, day, and year on one line" << endl;
    cin >> month >> day >> year;
    date2.Initialize(month, day, year);
    outFile << command << endl;
    switch (date.ComparedTo(date2))
    {
      case LESS : outFile << date.GetMonthAsString() << " "
                            <<date.GetDay() << ", "<<date.GetYear();
                  outFile << " comes before ";
            outFile << date2.GetMonthAsString() << " "
                    << date2.GetDay() << ", "<<date2.GetYear() << endl;
                  break;
    case GREATER:  outFile << date2.GetMonthAsString() << " "
                           << date2.GetDay() << ", "
                 << date2.GetYear();
                   outFile << " comes before ";
             outFile << date.GetMonthAsString() << " "
                    << date.GetDay() << ", "
                << date.GetYear() << endl;
                  break;

    case EQUAL  : outFile << date.GetMonthAsString() << " "
                          << date.GetDay() << ", "
                << date.GetYear();
            outFile << " and ";
            outFile << date2.GetMonthAsString() << " "
                     << date2.GetDay() << ", "
                 << date2.GetYear() << endl;
            outFile << " are equal " << endl;
                  break;
    }
  }
  else if ("DaysAway")
  {
    cout << "Input days away" << endl;
    cin >> daysAway;
    date2 = date.Adjust(daysAway);
    outFile << command << endl;
    outFile << date.GetMonthAsString() << " " << date.GetDay() << ", "
          << date.GetYear();
    outFile << " plus " << daysAway << " is ";
    outFile << date2.GetMonthAsString() << " " << date2.GetDay() << ", "
        << date2.GetYear() << endl;
  }
  else
    cout << "Unrecognized command." << endl;
  cout << "Input a command or Quit to terminate the test" << endl;
  cin >> command;
  }

  outFile.close();
}
```