ECS40 Spring 2018 2018-05-28

Homework 4 due Thu 2018-06-07 at 23:59

Use the handin directory hw4 to submit your work

Airports and Runways

Description

In this assignment (HW4), you will implement two programs that can be used to plan a flight. The first program called **`airport`** prints a detailed description of an airport including the names and lengths of its runways. The second program called **`distance`** computes and prints the distance between two airports. In a fictional scenario, a pilot who plans a trip could check the details of available runways at a given airport and compute the distances between airports.

This assignment will test your C++ proficiency in opening and reading text files, using strings, using STL containers, using STL algorithms, and implementing function objects.

Program specifications

airport

The **`airport`** program prints a description of an airport. It should read data from two text files containing information about Federal Aviation Administration (FAA) facilities and runways. The files **`Facilities.txt`** and **`Runways.txt`** are provided and contain a list of 19700 facilities (such as airports, heliports, seaplane bases) and 23595 runways located mostly in the United States, but also in remote locations. Each line in the **`Facilities.txt`** file contains the description of a facility (airport, heliport or seaplane base). The first part of the line contains the site number, a 10-character string that uniquely identifies the facility. The rest of the line contains other information, including the facility type, name, code, and position (latitude and longitude in various formats). For example, San Francisco International Airport has site number **`02187.*A`** , type **`AIRPORT`** , code **`SFO`**, name **`SAN FRANCISCO INTL`** , latitude **`135427.7000N`** and longitude **`440551.5000W`** (expressed in seconds decimal). The positions of these fields in the line are:

- Site number:	characters 1-10
- Type:	characters 12-24
- Code:	characters 25-28
- Name:	characters 131-180
- Latitude (sec decimal):	characters 536-547
- Longitude (sec decimal):	characters 563-574

Other fields are not relevant to this assignment.

Each line in the **`Runways.txt`** file contains the description of a runway. The first part of the line contains the site number of the facility it belongs to (i.e. the 10-character string described above). The rest of the line contains other information about the runway, including its name and length (in feet). For example, runway 10L/28R of San Francisco International airport has site number **`02187.*A`** , name **`10L/28R`** and a length of **`11870`** ft. The positions of these fields in the line are:

- Site number:	characters 1-10
- Name:	characters 14-20
- Length:	characters 21-25

Other fields are not relevant to this assignment.

The `airport` program should read the code of an airport (e.g. `SFO`) as a command line argument. The program should then open and read the above text files, create appropriate `Facility` and `Runway` objects, and store them using STL vectors. The STL `find_if` algorithm should be used together with a function object `Code` defined in the file `Code.h` to find the airport having the given code. The program should then print the description of the airport (site number, code and name). The program should then search for all runways that belong to the identified airport. This search will use the STL `stable_partition` algorithm and a function object `SiteNumber` defined in the file `SiteNumber.h` . The program should then print a description of all the runways of the airport (site number, name and length).

Notes:

- The airport code entered as a command line argument may consist of 3 or 4 characters. The program should be able to process both cases correctly.
- If the airport code is not found (e.g. `ABCD`), the program should print
  `ABCD not found`
  and exit.
- If the airport code given is longer than 4 characters, the program should print
  `Airport code must be at most 4 characters`
  and exit.

You are given the files `Facility.h` and `Runway.h` which declare classes that represent facilities and runways respectively. You should not modify these files. You must create the files `Facility.cpp` and `Runway.cpp` to implement the member functions of the classes `Facility` and `Runway` . You are also given files `testFacility.cpp` and `testRunway.cpp`, which are used to test your implementation of the `Facility` and the `Runway` objects.
You must create the files `Code.h` and `SiteNumber.h` to define the corresponding function objects. The programs `testCode.cpp` and `testSiteNumber.cpp` are provided and should not be modified. You are also given the files `gcdistance.h` and `gcdistance.cpp` that include the implementation of the calculation of the distance between two locations on Earth specified by their latitudes and longitudes. These two files should not be modified.

distance

The `distance` program prints the distance between two airports. The airport codes are read from the command line arguments. The distance is printed in nautical miles, rounded to an integer. See the example output files for the output format.

Notes:

- The airport codes are entered as command line arguments and may consist of 3 or 4 characters. The program should be able to process both cases correctly.

- If any of the airports entered is not found (e.g. **ABCD**), the program should print
  `ABCD not found`
  and exit.
- If any of the airport codes given is longer than 4 characters, the program should print
  `Airport code must be at most 4 characters`
  and exit.

Description of the Facility and Runway classes

Facility class

The member functions of the **`Facility`** class are defined as follows:

`Facility(string s)`
The constructor takes a single **`string`** argument. The argument **`s`** contains a full line read from the **`Facilities.txt`** file. The constructor should initialize the data members of **`Facility`** by selecting the appropriate substrings from the argument. The latitude and longitude fields should be converted to double values using the **`convert_latitude`** and **`convert_longitude`** member functions. The sign of the **`latitude_`** and **`longitude_`** data members should be determined by checking whether the latitude and longitude fields end with **`N`** or **`S,`** and **`E`** or **`W`** respectively.

`string site_number(void) const`
This function returns the facility's site number.

`string type(void) const`
This function returns the facility's type.

`string code(void) const`
This function returns the facility's code.

`string name(void) const`
This function returns the facility's name.

`double latitude(void) const`
This function returns the latitude of the facility in degrees decimal. Latitudes in the southern hemisphere are negative numbers.

`double longitude(void) const`
This function returns the longitude of the facility in degrees decimal. Longitudes in the western hemisphere are negative numbers.

`double distance(double lat, double lon) const`
This function returns the distance in nautical miles between the facility and the position defined by (lat,lon) in degrees decimal. The implementation of this function uses the **`gcdistance`** function provided in files **`gcdistance.h`** and **`gcdistance.cpp`** .

`double convert_latitude(string s) const`

This function converts the string **`s`** representing a latitude in seconds decimal to a **`double`** value in degrees decimal. One degree is 3600 seconds. The sign of the result is positive if the string **`s`** ends with **`N`** and negative if it ends with **`S`** . For example, the latitude represented by the string **`135427.7000N`** should be converted to the value **`37.6188`**

`double convert_longitude(string s) const`
This function converts the string **`s`** representing a longitude in seconds decimal to a **`double`** value in degrees decimal. One degree is 3600 seconds. The sign of the result is positive if the string **`s`** ends with **`E`** and negative if it ends with **`W`** . For example, the longitude represented by the string **`440551.5000W`** should be converted to the value **`-122.3754`** .

Runway class

The member functions of the **`Runway`** class are defined as follows:

`Runway(string s)`
The constructor takes a single **`string`** argument. The argument **`s`** contains a full line read from the **`Runways.txt`** file. The constructor should initialize the data members of **`Runway`** by selecting the appropriate substrings from the argument.

`string site_number(void) const`
This function returns the site number of the facility that the runway belongs to.

`string name(void) const`
This function returns the name of the runway.

`int length(void) const`
This function returns the length of the runway in ft.

`int convert_length(string s) const`
This function converts the string **`s`** representing a runway length to an **`int`** value in feet.

Function objects

Two function objects must be defined. The **`Code`** function object (to be defined in the file **`Code.h`**) is used to identify facilities having a given code. The **`SiteNumber`** function object (to be defined in the file **`SiteNumber.h`**) is used to identify either a **`Facility`** or a **`Runway`** having a given site number. Note that this requires the use of a template in the implementation of the **`SiteNumber`** function object. The programs **`testCode.cpp`** and **`testSiteNumber.cpp`** are provided to test the function objects, and should not be modified.

Test programs

The test programs **`testFacility.cpp`**, **`testRunway.cpp`**, **`testCode.cpp`** and **`testSiteNumber.cpp`** are provided and should not be modified. These programs will be used (with the corresponding input and output files) to test your implementations of the **`Facility`** and **`Runway`** classes and of the **`Code`** and **`SiteNumber`** function objects.

HW4 Assignment

Your task is to implement the files **`Facility.cpp`** , **`Runway.cpp`** , **`Code.h`** , **`SiteNumber.h`** , **`airport.cpp`** and **`distance.cpp`**. All programs should build without warning on CSIF using the command

```
$ make
```

Do not use C++11 or C++14 features in your source files. Use the **`g++`** compiler with the **`–Wall`** option.

Test cases

Five test cases of the **`airport`** program are provided with corresponding output files. Shell script files named **`testairport1.sh`** to **`testairport5.sh`** are provided and include the invocation of the **`airport`** program with its command line arguments. The files **`testairport1.out`** to **`testairport5.out`** contain the corresponding output. Similar files are provided to test the **`distance`** program. Test output files for the programs **`testCode`**, **`testSiteNumber`**, **`testFacility`** and **`testRunway`** are also provided.

Examples of use

```
$ ./airport SFO
02187.*A   SFO  SAN FRANCISCO INTL
02187.*A   01L/19R 7650
02187.*A   01R/19L 8650
02187.*A   10L/28R 11870
02187.*A   10R/28L 11381
02187.*A   28X     0

$ ./airport SAN
02170.*A   SAN  SAN DIEGO INTL
02170.*A   09/27   9400

$ ./distance SAN SFO
SAN  - SFO  388 NM

$ ./distance SAN ORD
SAN  - ORD  1494 NM
```

Verify that your programs reproduce the tests *exactly*. Use the **`diff`** command to compare your output with the reference test output files. Note that other test files may also be used when grading your implementation.

Submission

Create a tar file named **`hw4.tar`** containing all the files needed to build the **`airport`**, **`distance`**, **`testFacility`**, **`testRunway`**, **`testCode`**, and **`testSiteNumber`** programs. In order to limit file size, do NOT include the files **`Facilities.txt`** and **`Runways.txt`** in your tar file. Submit your project using:

```
$ handin cs40 hw4 hw4.tar
```