**Person Class**

Discussion: what do we need to implement a Person class?

We’ll keep things really simple…

|  |
| --- |
| **Person** |
| - firstName: string  - lastName: string  - dob: OCCCDate |
| + Person (string firstName, string lastName)  + Person (string firstName, string lastName, OCCCDate dob)  + string getFirstName()  + string getLastName()  + OCCCDate getDate()  + void setFirstName(string fn)  + void setLastName(string ln)  + int getAgeInYears()  + bool equals(Person p)  - bool equalsIgnoreCase(string s1, string s2 ) // helper function for equals  - string toUpperCase(string s)  + String toString() // output “Lastname, Firstname (dob)” |

Note that we’ve made reference to another class, OCCCDate:

**OCCCDate Class**

|  |
| --- |
| **OCCCDate** |
| - dayOfMonth: int // 1..31  - monthOfYear: int // 1..12  - year: int // e.g. 2015  - bool dateFormat // defaults to true; if true use US date format;  // if false use European date format |
| + OCCCDate() // default constructor, uses current system date and time  + OCCCDate(int day, int month, int year) // as defined above  + int getDayofMonth() // 1 for the first, 2 for the second…  + int getMonth() // 1 for January, 2 for February…  + string getNameOfMonth() // January, February, March…  + int getYear() // 2013…  + int getDifference(OCCCDate d1) // elapsed time in years  + int getDifference() // elapsed time in years from current time  + void setDateFormat(bool) // sets date format  + bool equals(OCCCDate d) // compare only month, day, year  + string toString() // output formatted date string as mm/dd/yyyy or dd/mm/yyyy  // depending on value of dateFormat |

OCCCDate will need to make use of the built-in time and calendar functions of C++, so let’s look at a programming example:

// http://www.tutorialspoint.com/cplusplus/cpp\_date\_time.htm

#include <iostream>

#include <ctime>

using namespace std;

int main( )

{

// current date/time based on current system

time\_t now = time(0);

cout << "Number of sec since January 1,1970:" << now << endl;

tm \*ltm = localtime(&now);

// print various components of tm structure.

cout << "Year: "<< 1900 + ltm->tm\_year << endl;

cout << "Month: "<< 1 + ltm->tm\_mon<< endl;

cout << "Day: "<< ltm->tm\_mday << endl;

cout << "Time: "<< 1 + ltm->tm\_hour << ":";

cout << 1 + ltm->tm\_min << ":";

cout << 1 + ltm->tm\_sec << endl;

}

Enter this program (or copy and paste from the provided URL), compile, and execute.

As you can see, this makes use of some language features that we haven’t yet seen,

such as the address-of operator & and the operator -> operator for dereferencing a structure.

We don’t need to worry about what they mean or how they work (yet) – we just want to make use of the C++ time facilities to implement our own, simpler time class while hiding the complexity of the ctime library.

The copy constructor isn’t actually needed; C++ gives us one by default that does a simple member-wise copy of the source object. Since we don’t use pointers or allocate memory dynamically in our construtors, that’s good enough. If we wanted to write it anyway, it would look like this. Note the address-of operator. Since we are within the class we are allowed direct access of the private data fields.

OCCCDate (const OCCCDate& d){

this.dayOfMonth = d.dayOfMonth;

this.monthOfYear = d.monthOfYear;

this.year = d.year;

}

Now we are ready to start writing the OCCCDate class. The basic framework of a C++ class looks like this:

class NameOfClass{

private: // private stuff goes here

public: // public stuff goes here

protected: // private stuff goes here

}; // note the trailing semicolon, it’s important!

Now let’s fill in the fields specified by the class diagram and wrap it up in a header file:

**// file OCCCDate.h**

**#ifndef OCCCDATE\_H**

**#define OCCCDATE\_H**

class OCCCDate{

private:

int dayOfMonth;

int monthOfYear;

int year;

bool dateFormat;

public:

OCCCDate();

OCCCDate(int day, int month, int year);

int getDayofMonth();

int getMonth();

string getNameOfMonth();

int getYear();

int getDifference(OCCCDate d1, OCCCDate d2);

int getDifference(OCCCDate d);

void setDateFormat(bool df);

bool equals(OCCCDate d);

string toString();

};

**#endif**

The preprocessor directives ensure that if our code “includes” OCCCDate.h multiple times it only gets processed once.

The actual implementation goes in OCCCDate.cpp as follows:

// file OCCCDate.cpp

#include "OCCCDate.h"

OCCCDate::OCCCDate(){

// code goes here

}

OCCCDate::OCCCDate(int day, int month, int year){

// code goes here

}

// and so on for all of the class functions

Let’s take a few moments and implement these methods. The only tricky one is the default constructor, which needs to pull the current date and time from the system, but we have sample code that shows us how to do that. To test them we need a third program that makes use of the OCCCDate object…

// file testOCCCDate.cpp

#include<iostream>

#include "OCCCDate.h"

using namespace std;

int main(){

OCCCDate d1;

OCCCDate d2(12,25,2013);

cout << d1.getMonth() << endl;

// …and so on

return EXIT\_SUCCESS;

}

So our program contains three files: a header file (interface and specifications), a class .cpp file (implementation), and a main program that makes use of the object we have constructed. In the real world the header file is provided to the client as is, and the .cpp code as a precompiled binary file (we don’t show the clients the implementation details).

Once we have tested our OCCCDate class, we can now implement and test our Person class.