

Computer Science II — CSci 1200

Lecture 3 — Classes, Part 1

Review from Lecture 2

- C++ strings from the standard library:
 - hold sequences of characters,
 - have a sophisticated set of operations defined on them, including resizing, changing characters, and appending new characters
- Vectors, also from the standard library:
 - effectively they are smart, dynamically-sized arrays
 - should (almost) always be used instead of arrays
- Strings and vectors:
 - grow and shrink from their end (highest subscripts) rather than their beginning (subscript 0),
 - should (almost) always be passed by reference; when it is desirable to ensure that no changes are made, they should be passed by constant reference
- Recursion is a way of defining a function and or a structure in terms of simpler instances of itself. We will see examples of recursive definitions and recursive functions throughout the semester.

Today's Lecture: C++ classes

- Types and defining new types
- A `Date` class.
- Class declaration: member variables and member functions
- Using the class member functions
- Class scope
- Member function implementation
- Classes vs. structs
- Designing classes

Types and Defining New Types

- Q: What is a type?
A: A structuring of memory plus a set of operations (functions) that can be applied to that structured memory.
- Examples: integers, doubles, strings, and vectors.

- In many cases, when we are using a class we do not know how that memory is structured. Instead, what we really think about is the set of operations (functions) that can be applied.
- To clarify, let's focus on strings and vectors. These are classes. We'll outline what we know about:
 - The structure of memory within each class object
 - The set of operations defined
- We are now ready to start defining our own new types using classes.

Example: A Date Class

- Many programs require information about dates.
- Information stored about the date includes the month, the day and the year.
- Operations on the date include recording it, printing it, asking if two dates are equal, flipping over to the next day (incrementing), etc.

C++ Classes

- A C++ class consists of
 - a collection of member variables, usually `private`, and
 - a collection of member functions, usually `public`, which operate on these variables.
- `public` member functions can be accessed directly from outside the class,
- `private` member functions and member variables can only be accessed indirectly, through public member functions.
- We will look at the example of the `Date` class declaration.

Using C++ classes

We have been using C++ classes (from the standard library) already this semester, so by studying how the `Date` example we are just repeating what we have already done, but this time with a class we have defined. We will illustrate with the file `date_main.cpp`:

- Defining class objects.
 - **Important note:** each object we create of type `Date` has its own distinct member variables.
- Calling class member functions for class objects using the “dot” notation. For example,

```
tomorrow.increment();
```

- Note: after looking at the example, we still will not have examined how the class member functions are implemented! This is an important feature of class design.

Exercise

Hand-write code that could be added to `date_main.cpp` to read in another date, check if it is a leap-year, and check if it is equal to `tomorrow`. In each of the latter two cases, output an appropriate message based on the results of the check.

Class Implementation

We will illustrate the concepts of class implementation through the `Date` example, looking at the implementation file `Date.cpp`

Class scope notation

- `Date::` indicates that what follows is within the scope of the class.
- You will have to get accustomed to the notation of class member functions
- Within class scope, the member functions and member variables are accessible without the name of the object.

Constructors

These are special functions that initialize the values of the member variables. You have already used constructors for string and vector objects.

- The syntax of the call to the constructor can be confusing. It mixes variable definitions and function calls. (See `date_main.cpp`)
- “Default constructors” have no arguments.
- Multiple constructors are allowed, just like multiple functions with the same name are allowed.
- The compiler determines which one to call based on the argument list. This must match (in type) the parameter list, just like any other function call.
- When a new object is created, **EXACTLY** one constructor for the object is called. You can’t prevent it from happening. You can only control which one is called.

Member Functions

These are like ordinary functions except:

- They can access and modify the object’s member variables.
- They can call the member functions without using an object name.
- Their syntax is slightly different because they are defined within class scope.

Important member functions to study

- `set` and `get` functions access and change a day, month or year.
- `increment` uses another member function, `isLastDayInMonth`.
- `isEqual` accepts a second `Date` object and then accesses its values directly using the dot notation.
 - Since we are inside class `Date` scope, this is allowed.
 - The name of the second object, `date2`, is required to indicate that we are interested in its member variables.
- `lastDayInMonth` uses the `const` array defined at the start of the `.cpp` file.

Accessing and changing member variables

- When the member variables are private, the only means of accessing them and changing them from outside the class is through member functions.
- If member variables are made public, they can be accessed directly. This is usually considered bad style and will not be allowed in this course.

Non-member functions

- Functions that are not members must interact with `Date` objects through the class public members (the “public interface” declared for the class).
- These may perform operations that are closely associated with the class and therefore “feel” like member functions.
- One example is the function `sameDay` which accepts two `Date` objects and compares them by accessing their day and month values through their public member functions.

Header files (.h) and Implementation files (.cpp)

The code for the `Date` example is in three files:

- `Date.h` contains the class declaration.
- `Date.cpp` contains the member function definitions. `Date.h` is included.
- `main_date.cpp` contains the code outside the class. `Date.h` again is included.
- The files `Date.cpp` and `main_date.cpp` are compiled separately and linked to form the executable program.
- Different organizations of the code are possible, but not preferable.
 - For example, we could have combined `Date.h` and `Date.cpp` into a single file and included them in the main program.
 - In this case, we would not have to compile two separate files.
- In many large projects, programmers establish the convention that requires exactly two files per class, one header file and one implementation file.

Constant member functions

Member functions that do not change the member variables should be declared `const`

- For example,

```
bool Date::isEqual(const Date &date2) const
```
- This must appear consistently in **both** the member function declaration in the class declaration and the member function definition.
- `const` objects, usually passed as parameters, can **ONLY** use `const` member functions
 - Remember, you should only pass objects by value under special circumstances. In general, pass them by `const` reference if you do not want them to change.
- While you are learning, you will tend to make errors in determining which member functions should or should not be `const`. Watch for them when you compile files containing your class headers and member function definitions.

Exercise

Add a member function to the `Date` class to add a given number of days to the `Date` object. The number should be the only argument and it should be an unsigned int. Should this function be `const`?

Classes vs. structs

- Technically, a `struct` is a `class` where the default protection is `public`, not `private`.
 - As mentioned above, when a member variable is `public` it can be accessed and changed directly using the dot notation.
 - For example,

```
tomorrow.day = 52;
```

We can see immediately that this is dangerous because a day of 52 is invalid!
- The usual practice of using `struct` is all `public` members and no member functions. This practice is contrary to the usual conventions of C++ programming.

Rule for the duration of CS II: We will not allow the use of `struct`, and no member variable can be made `public`. This will be enforced with various levels of grading penalties.

C++ vs. Java Classes

- In C++, classes have sections labeled `public` and `private`, but there can be multiple `public` and `private` sections. In Java, each individual item is tagged `public` or `private`.
- Class declarations and class definitions are separated in C++, whereas they are together in Java.
- In C++ here is a semi-colon at the end of the class declaration (after the `}`).

Designing and implementing classes

This takes a lot of practice, but here are some ideas to start from:

- Begin by outlining what the class objects should be able to do. This gives a start on the member functions.
- Outline what member variables might be needed to accomplish this.
- Write a draft class declaration in a .h file.
- Write code that uses the member functions. Revise the class .h file as necessary.
- Write the class .cpp file that implements the member functions

In general, do not be afraid of major rewrites if you find a class is not working correctly or is not as easy to use as you intended. This happens frequently in practice!