CPS222 Lecture: Course Introduction; Introduction to C++

last revised December 16, 2014

Objectives:

1. To introduce the study of data structures and algorithms as "classic" Computer Science
2. To introduce this course
3. To introduce C++

Materials:

1. Syllabus
2. C++ background questionnaire
3. Java/C++ Comparison handout
4. Creating and Compiling C++ Programs handout

I. Course Introduction

A. The study of data types and structures, and the related topic of algorithms, has long been one of the core areas of a CS curriculum.

1. As a result, CS has developed a repertoire of "standard" data structures and "standard" algorithms with which every well-educated computer scientist should be familiar.

2. In a sense, the study of data structures and algorithms can be thought of as being to computer science what anatomy is to a field like biology or medicine.

B. Thus, what you will be studying in this course can be thought of as "classic Computer Science".

1. Historically, data structures have been studied fairly early in the CS curriculum - typically in the second course - with perhaps more advanced work in an upper-level course.

a) For example, in the second CS course in the freshman year in the curriculum we followed until 1999-2000, students would learn about both using and implementing data structures at the same time. (E.g. the unit on linked lists in CS122 talked about both how to use lists and about how to implement them.)

b) This was motivated by the need to use various data structures in more advanced courses.

2. In our current curriculum at Gordon, we have separated the study of *using* data structures from the formal study of data structures and their implementation. This reflects the fact that well developed software libraries (like the Java API and the C++ Standard Template Library) provide important data structures that can be used without understanding the details how how they are implemented.

   Example: In CPS122 we teach the *use* of various collections (`Sets`, `Lists`, `Maps`) without talking about *how* they are implemented. In this course, we will learn about *how* these structures, and others, are implemented.

C. Nonetheless, it is important at some point in one's CS education to study "classic" data structures and algorithms material - which is the subject of this course. There a number of reasons why this is very important:

1. Intelligent use of the standard libraries like the Java collections framework or the C++ Standard Template Library requires some understanding of what's going on "under the hood". For example, we've already seen that if you need a List in Java you can choose to use either an `ArrayList` or a `LinkedList`; when using a `Set`, you can choose to use either a `TreeSet` or a `HashSet`; and when using a `Map` you can choose to use either a `TreeMap` or a `HashMap`. Making an intelligent choice requires some understanding of how these different implementations are actually realized.

2. There will be many problems for which the data structures in the standard library don't really meet the need. For example, one structure that is widely useful in many kinds of problems is the *graph*. There is no graph structure per se in the Java API or the C++ STL - so you will have to create your own if you need to solve such problems. (In fact, the different problems are best solved using very different structures, which is why it would be hard to define a single standard structure analogous to `Sets`, `Lists`, `Maps`.)

3. Someone has to create the standard libraries that we depend on.

D. One reason why the study of data structures is important is that a given problem that could be very complex to solve can be made very simple by finding the right representation (data structure) or algorithm.

   EXAMPLE: Suppose we wanted to maintain a list of pairs consisting of an ASCII code and its corresponding character - e.g. 65 would be associated with 'A' etc.

   1. We could use an array of objects in Java (or tuples in Python), with each object/tuple containing a code and its associated character. In this case we would need to search through the array every time we wanted to look up the character associated with a given code.

   2. Or we could use a Java map (or a Python dictionary), with codes as keys and corresponding characters as values - in which case we would use get() in Java (or [] in Python) to lookup the character corresponding to a given code.

   3. Or we could use a single array of characters, with the element at position i being the character whose code is i. (Here we take advantage of the fact that the ASCII codes are small non-negative integers.) In this case, we would find the character corresponding to a given code by just using the code as an index into the array.

4. But the simplest way of all would be to take advantage of the fact that characters are represented internally by their codes - so converting a code to a character can be done by just a type cast - with no data structure needed at all!

E. Another important concept in the course is the notion of analysis of algorithms. When solving a problem, a good choice of algorithm can make a tremendous difference in performance - but often what one has to do is to look at various alternative ways of solving a particular problem, analyze the algorithms involved for the particular problem, and then choose the best approach. (This is where some of the things you studied in Discrete Math come in to play!)

F. Go over syllabus.

II. Introduction to C++

A. As noted in the syllabus, we will be using C++ in this course, though learning C++ is a secondary objective of the course.

1. Most of your learning in this regard will take place through projects and lab

2. Distribute, then collect C++ background questionnaire

3. Discuss philosophy of having less experienced people work with less experienced people etc.

B. Distribute Java / C++ Comparison HO

Go over sections 1-9

C. Distribute Creating/Compiling C++ Handout

Go over