ProgrammingAbstractions-Lecture01

Instructor (Julie Zelenski): Hi. Welcome to CS106B, programming [inaudible]. The website was probably the most important thing to take away from here, right, is where can you find information about the class? We're going to talk today and give some overview and stuff like that, but this is kind of the home base for all the material. If you managed to get the handouts on the way in, you're golden. Otherwise, you can grab them from the website. There's a lot of background information, staff information, office hours, all sorts of stuff gathered there. It's kind of one stop shopping for figuring things out about our course.

Let me tell you what I'm going to do today. The first day, I know a lot of you are shopping and trying to figure out what's the right fit for you, so hopefully today, I'm going to try to give you some information that will help you make a good decision. I want to tell you a little bit of what our course is about, tell you about the administration and logistics – most of that stuff is pretty ordinary, and you can read about it. Of course, I have to do a little bit of marketing. I get to give you my unbiased opinions of why this is the absolute best class you could possibly take.

I get paid per student, you know. That's not true. Maybe we'll even have time to check out a little bit of the C++ language before we're done. Let me tell you about CS106. CS106 is the introductory programming sequence here at Stanford. It's our version of CS1, where you start at the beginning when you are interested in learning more about programming. We have a two-quarter sequence, A and B, that kind of follow together. They're not particularly tightly coupled, which is to say if you took A and you took a break, you could come back to B.

We also have an alternate form of B, the CS106X, which is kind of an honors version of that second course. So after you've taken that first course and you're feeling really jazzed up, there is an alternative more intense way to get through the second course that's offered. It's offered this quarter, in fact, so if you're thinking about that, that is an option for you as well as sticking with us.

What do we do in 106A? The official title is programming methodology. It is starting at the very beginning and assuming you have no background in programming. It's teaching you how it works, what the languages look like, what the syntax is like, what things you need to know about how to solve problems using a computer. It covers a lot of the fundamentals about logic control and in general, I think the big issues of 106A have less to do with any particular syntax or feature that you learn about a language but about how do you solve problems on a computer?

Somebody gives you a specification of you need to write a program that does X, and you have to figure out how to make that happen, how to break it down, how to step through it, how to develop it, how to test it, how to iterate on it, how to make changes in it later, how to debug it when it's not working well, and those things, I think, transcend any particular language. W e happen to use the Java programming language in our 106A course because it’s a great tool for introductory programming, but I don’t think of that as being really critical.

In fact, if your introductory course was in some other fairly modern, high level language – let’s say you learned in Python, C, Scheme, or something other than Java, you’re probably still fine, ‘cause what we’re really counting on in B is that you know how to program and think like a computer scientist. Not a lot of the details of the syntax are going to be important to us. B picks up from there, saying okay, you’ve got some fundamentals. Let’s start really learning some of the techniques that extend the range of problems you can solve.

We look at recursion, which is one of the fundamental problem solving techniques that involves using something akin to mathematical induction to solve problems in terms of themselves. Looking at a lot of algorithms for sorting and searching and hashing and doing things efficiently, knowing how to compare and contrast alternatives in algorithms, having some formalisms by which to discuss those things, and learn some of the classics that are out there for solving these problems.

The dynamic data structure, which involves use of pointers to construct things like lists, trees and heaps, graphs that model certain structures that are very effective in solving certain kinds of problems – we’ll work through those. A large part of our time is spent on this concept of data extraction, which is why extraction shows up as the main word in our title. As we start to solve more and more complex problems, the code itself can kind of become overwhelming if we don’t have some technique for managing that complexity.

One of the big themes for 106B is how we can use this idea of extraction, building something and dealing with the low level details but then closing up the box and treating it as kind of a fixed entity and building on top of that and then closing another layer around that as a way of working on something, finishing it and moving on to some larger piece without having those details cloud our way. It’s a very powerful technique for solving larger problems.

In that context, we’ll be looking at some of the classic data structures, like stacks, cubes, lists, maps and sets as part of the domain for that. We do happen to use the C++ programming language, but this is not a C++ course, so to be clear about what you’re getting versus what you wanted, we use Java here. We use C++ here. We happen to think they’re good reasons to actually expose you to both languages. In particular, C++ is an enormous language. It has a lot of language features as well as a very large standard library, and our goal is not at all to turn you into this industrial strength knows every detail about quirks and ins and outs of C++.

There is another class, 193D, that does attempt to do that. In fact, if that’s what you’re looking for, I suggest you take a look at that. What we’re here about is learning advanced programming techniques – taking those foundations and building on them to be able to solve more interesting problems. We happen to use C++. You will learn some C++, but I almost consider it a side effect of what we’re doing.

Just a little note on placement – if you're kind of in between and not really sure, these are the very rough guidelines, but they give you some idea of which groups gravitate where. If you are new to programming or you're not confident about your background – maybe it was a long time ago. Maybe it was self-taught. Maybe it was in a course that you felt was not as good as it could have been or you didn't do as well in it, 106A is a great place to start. It actually is by all accounts an extremely popular course at Stanford and services a wide group of people with a little bit of background or no background.

If you do have something like a solid first course experience – you did well in 106A or took a similar course or perhaps even self-taught your way through a lot of those materials and you feel ready to move on, 106B. An AP course in high school – the A curriculum is a pretty good match for 106A here, so you're in a great place.

If you have this and you've got a little bit more going for you or you're super enthused and you have a lot of extra time this quarter and want to sit in the company of only the uber geeks, you can check out 106X, which covers the same kind of topical ground but at a different level intensity. It amps up a little bit of the assignments, covers some of the material that we won't get a chance to cover, and just pushes the envelope a little bit there.

If you have experience comparable to the first two courses – you've done all the things that we're talking about here in B and you feel comfortable with it, it might be that the right place for you is 107, which is the third course in our sequence. That's somewhat rare, so if you're thinking about that, I encourage you to talk to me a little bit to make sure that you won't be missing out on something important in doing so, but certainly there are students who have – for example, the APCSAD curriculum is pretty comparable to this course here, and so depending on how high quality the course you had was, it might very well be that 107 is right.

In some situations where that course was a little bit lacking, there may be some ways that we can help reinforce the things you've learned and build a stronger foundation to move forward from rather than jumping ahead. Any questions about placement?

Let's talk philosophy. I think there's a statement about what we officially are, but I also think that 106 has a long tradition at Stanford that comes back from student motivation, which is interesting. I was here as an undergrad in the 80s when the 106s were just getting off the ground, and at the time, Stanford only had a graduate computer science department, and the belief in the ancient period for computer science was you should get a math degree, and only then would you be mature enough to learn about computers.

There was a groundswell of Stanford students who said we want access to programming. We want it. Part of the 106 was a really careful thought about what the 106 would be a Stanford and what we want them to be in a philosophical sense. One is that we welcome students of all majors and backgrounds. We don't have a version of 106 that's for the majors or potential majors and a version that's for the non-majors and a version that's for terminals. We really think that we can bring you all together and design a course that addresses this wide disparate group but still serves it well.

I'm going to turn you into a CS major. That's my plan. At Stanford right now, not having to make that choice about a major until junior year is a gift to allow you to explore and to feel unencumbered by having made some decision when you applied, and I think it's important to respect that gift that Stanford gave you by trying to make sure our courses don't funnel you one way or the other before you figure it out. You are all welcome here. We try to make it accessible to everyone. We have certain plans that help to make that work.

We do try to provide a solid, practical foundation in programming that given our placement at Stanford in the middle of the Silicon Valley, there's kind of a strong influence for us to try to produce students who from the get go are learning things that are actually quite useful outside of the classroom rather than teach you a very academic and mathematical language like Scheme that is very rarely used outside of the classroom. We're trying to teach you on the tools, languages and techniques that are actually in active practice.

We are using Java and C++, two of the most prevalent languages out in the industry, and we do a lot of learn by doing. We assign challenging, full-fledged programs that you work on and you build, and so it's not designed to be academic exercises. You really are building skills that have applicability here and outside of the class. We have a big emphasis on truth and beauty. This is one area in which some of the substitute courses that we have seen students come in with have a little bit more trouble with is tackling this part of it, which is that there are a lot of ways you can get a program to work. Many of them are not pretty.

You can just type and type and type and eventually, you can get your way to something that works. That, in the end, might produce a program that from external appearances works. It plays hangman or whatever was the desired goal, but that internally is a mess. It's not well structured. It's not well [inaudible]. It's not easy to understand. It wouldn't be easy to modify. It makes a lot of decision that are really sub optimal, and we're really interested in producing engineers that have a good sense of design and really appreciate what is involved in writing good, well designed software, not just working software.

We will be giving you feedback on both the correctness and functionality of your code but as important if not more so, also on how well we think you did at designing and implementing and writing code that is of a high quality. We make a big deal out of that. That is something that is not always shared by other classes, and so in particular, someone who is self taught or in a class that didn't emphasize this might feel that there's a little bit of a gap there where we need to make that up with you, and we can work with you.

This is a very individual thing, because there's not one good example of the perfect style in the way that a lot of different people express themselves in written communication very well but differently. The same thing is true about programs. You will have your own unique style, and we'll work with you to coach you on getting your style to come through and be beautiful and elegant.

This kind of comes back to point one as well is that we make heavy use of undergraduate section leaders as mentors in this program. We have a staff of 50 or so undergraduates who work with the 106A, B and X courses as a team. They have specific responsibilities with their section, so mentoring and grading and meeting with sections to give individualized feedback on their programs as well as answering questions, solving problems, being in the lair 30 plus hours.

They're most weekday nights about six hours and often well past the midnight when it's supposed to cut off solving people's problems, helping when you get stuck, answering your questions and making sure you're all making forward progress. That comes back to point one. In some universities where they're choosing to use their intro course as a weeder, it's like let's separate the weak from the [inaudible] early and let's make it really hard and not provide too much support, and that way we'll make sure we get the people we want.

We have a different idea. Programming is hard, especially when you're learning. There's a lot of complexity to master, and there are a lot of details that can interfere with moving forward, and we don't want you to get stuck on something that we can very easily resolve for you. So make sure you have accessible staff members in person, in email and regularly in session to help get through the roadblocks and keep you moving forward.

What do you need to know to be sure you're going to do well? There are people who think to be a good computer programmer, you need to be good at math, logic, and drinking Jolt. I think it comes down to more personality traits than any particular technique or skill. You don't need to be good at math. How much calculus and trig shows up in this? Not very much. A little bit of logic – that helps. I think it comes down to traits like curiosity and determination and hard work. Starting early, asking questions when you don't understand something, trying to solve the problem by logic – why does this case work and why does that case not work?

I think these are the skills that serve you the best in this class and probably every other thing you'd want to tackle. There is a lot of time that will be spent here to master this. I can talk about programming all I want, and you can go oh, yeah, that makes a lot of sense. But when you go to write it yourself, it's a very different experience. That's where being focused and staying on task and getting help when you get stuck can help you move through that.

My unbiased opinion about why 106B is one of the best courses at Stanford – it's going to be totally obvious when I say these things, and you guys are going to have to go along with it. I'm actually a big time geek, and I happen to love programming. That's why I'm a perfect fit for teaching this course. I have taught 106 B or X more than I've taught any other class that I've taught at Stanford in my time here, and that's because each quarter when we’re setting the schedule, I say give me B. Give me X. There’s no better course to teach.

Programming is just awesome. If you love programming, I think there’s almost nothing better to do in the world. You have this task. You’re trying to get there. You’re coding. You’re making stuff happen. You’re testing, iterating and running. You see stuff. You build things. When you’re done, you know it works. When a program works, you know it.

It does what it’s supposed to do. It gets the right answer. It plays the game. It solves the problem. Finding and fixing that last bug – although debugging is one of the last aspects of programming a lot of people bemoan about, I happen to think that if you are driven by debugging, that is one of the most awesome detective stories ever.

Trying to figure out why it happened when you did this, why it went this way when you moved that and what this effect caused and understanding once you make the fix how it fixes it – staying up late. I have stayed up late more nights debugging than anything else in my life, and I’m not sad about that at all. That means I’m in the right place. Hopefully, some of that resonates with you. If that sounds really awful to you, hopefully we can change your mind a bit. That is, in some ways, part of what drives computer science is this wanting to build things.

We are engineers. We have this computer science name. Just remember – any subject that’s name is something science is not a science. We’re trying to puff ourselves up a bit. We do a lot of really great work, and there are a lot of neat scientific principles and theories that underpin what we do, but in the end, I think what drives a lot of us is just the engineering – building stuff that is really neat.

It kind of reminds me that my husband’s a mechanical engineer, and so I used to be envious, because he would always build things. He’s always toting foam core around the campus on their paper bicycles and stuff, and it’s like they build all these things. Now, having watched all the things he builds, it’s like [inaudible] really hard. You need all these materials. You need all these tools. In computer science, you don’t need anything. You need a compiler and you need a computer. You need your thought. It’s like an abstraction we built out of our brains.

There’s this relatively small set of things that you need to master and then you can combine them in these very sophisticated and interesting ways to solve all sorts of problems. There’s a very low overhead, and the range of things you can attack with the same set of skills is huge. Every domain out there can benefit from somebody applying computers in a useful way without fail. There’s all sorts of problems where technology is part of the answer – not the only answer, but certainly something that you can take whatever interest you have and combine it with computer science and construct something cool.

I happen to think that what happens in the second course is amazing. The first course, you've kind of got to get up to speed, and there's a lot of basic material that needs to get covered, and it does set you on the right path, but in this course, we really get to blossom beyond the basic things.

There are a bunch of really neat and very accessible techniques that a second quarter student can understand and master and do really cool things with. You can learn how to do something like create a database that has a million entries and then ask for somebody by name and be able to instantaneously be able to find the name. Change the size of it. Make it ten million, a billion, and still be able to provide that kind of instantaneous access. You're going to learn how to do that.

The technique is not some superhuman thing. It's something very clever, admittedly, but it's very accessible. Taking that same million thing and learning how to sort it efficiently. What if you happen to know things about how it's almost sorted but just a little but out of sorts? Are there ways you can actually make it even faster to put it in sorted order? There are techniques, for example, like recursion that take on problems that you might not have any idea when you first look at the problem how to solve, but once you've got your head around recursion, you can look at that and say I can write a five line function that will solve that problem.

This is the kind of power we're going to give you with our quarter together. There's a bunch of really amazing theoretical and algorithmic stuff to explore that really increases the kind of things you can do with programming. I'm particularly fond of it. As always, I think the section leading program that we have created and built at Stanford is a huge part of what makes our 106 courses so successful, and so having somebody who's individually working with you, meeting with you weekly and giving you that feedback that's individualized and personalized for you and helping you get through the rough spots is a lot of what helps to make the experience very fun and very personal, too.

What section leaders do I have here? Not a one. Where are those section leaders? We haven't yet identified who's going to what class, so maybe they're all thinking they're going to go somewhere else, but they're wrong. We'll get some. 106B – great. You agree? Have I convinced you? Is anybody still hesitant? Is it a lot of work? Oh, no.

Let me tell you a little bit about logistics. There are some random things you may want to know about how the class works. We're going to meet here Monday, Wednesday, Friday 2:50 to 3:05. It looks like we almost exactly fit in our classroom, which means we're all going to be very friendly and cozy. The lectures are being taped and are available online, and so that has the neat side effect that you can watch them and review them later. You could watch them lots of times.

It also means that if it were pouring rain and you were sitting at home in your bunny slippers, you could just say hey, I'm not going outside and you could turn on your computer and watch. I'm a big fan of having you come in person. That's because I don't want to lecture to an empty room. I also think there's an interaction there that I'm fond of, and so I hope that you will make every effort to attend in person as much as you can. It is nice to know that if you do miss a lecture or you get caught up with something, you'll have a chance to review it later online.

We will have sections that will meet once a week, just like 106A. The section leader who meets with you is the one who will be grading your programs and sitting with you and doing the conferences. What you need to do to get yourself into a section – there are several section times listed. Ignore them.

What you do is you go to the online – if you go to our class webpage, there's a link that tells you how to sign up for a section. The section times aren't up right now, but they will go up tomorrow, and they'll be up through the weekend. You go in and put in your preferences about what times fit your schedule or not, and there's this big computer program that gets everybody into a section that fits into their schedule.

The signups will be up from Thursday at 5:00 until Sunday at 5:00. If by Sunday at 5:00 you've got your schedule fixed, then you're fine. If you happen to change after Sunday at 5:00 and after the assignments have been made, at that point, it's a little bit harder. We can make adjustments, but it's on a very case-by-case basis. The best thing you can do is by Sunday at 5:00 have a pretty firm idea about what you can and can't do. There's a list of preferences, so maybe what you can do is pick things that you know will work no matter what happens.

The workload – everybody wants to know how much work. It's a five-unit class, and it's a five unit engineering class. You get your five units worth, I would say, so don't worry about that. I won't shortchange you. We have programming assignments not quite weekly. I think there are seven of them across ten weeks, so you can calculate it out. It's about a week and a third for any particular one. The students report that it's about a 15-20 hour project, each of them. Some people get them done in less than that. Some people take a little bit more. I would say that's kind of the mode range for what people are reporting.

I do think that of the people who report less, some of those are people who are naturally gifted, but a lot of it has to do with your habits about how you work and how you make progress, so if you are one of the people who feels you might be more likely to be on the other end, you can come and talk to me and I can give you some suggestions.

Choosing to work in the lair where the helpers are on duty has really positive effects in that when you get stuck, you have easy access to somebody helping you work through it rather than spending an hour or two fighting something that turns out to be simple but required knowing something that you didn't yet know.

I'm a big fan of learning things yourself. There's also a time when a well-placed bit of advice from somebody can save you a lot of time. There will be a midterm and a final exam. They'll both be in class, open book, open notes. The midterm is actually technically out of class. We're going to have it at night because we need more than a 50-

minute period to get any coverage of that. Our final exam is scheduled in our university-scheduled slot. Sadly, that is at the very end of the exam week, but that’s when the registrar put us, and that’s when we need to go. You may want to take a look at that before you head off for spring break.

Unfortunately, our publisher will not allow me to distribute the course reader electronically. They’re not exactly very in the modern age on this. Yeah, I’m working on it is the truth. By the time the world sees this, hopefully, we will have some other strategy. We currently are in negotiations. The nice thing about the course reader is that we have not changed it in the last year. If you know somebody who has it from last fall or last spring, it has some minor edits and typos that were fixed, so if you can get a hold of an old one, it’s good.

We are getting no royalties on it. We’re publishing it at production cost, so where it would have been a $100.00 textbook had it been bound and snappy, you’re just getting what it cost to photocopy and bind the thing, and Eric and I are eating ramen. It’s hopefully cheap enough that you can find a way to get to one or get an old one without it being too much of an obstacle.

People in general find the course reader to be an asset. We do say it’s required reading. It does have a lot of material that’s very useful in understanding the course. There are other people who don’t make as big a use of it, but there are some sections that are really very valuable and other ones that may be more or less depending on your learning style. There’s also a lot of good sample problems and review questions in it that help to test your understanding.

You know, it’s tricky, because the university in general discourages you from giving alternate exams because of – you can imagine the issues of having an exam that has been seen by some number of students before some other number of students take it. Even though we’re all bound by the honor code, it does create a situation where there is some temptation. It’s possible it could be a little bit early without a lot of gap, but I don’t think early enough to make a lot of help is the truth. We can talk about it. The current plan is not, I would say.

A little bit about compilers. We use C++ and we also use some custom libraries, which limits us to distributing on a certain number of platforms we’ve had a good chance to test on and work with. The compilers that we have support for is X Code on the Macintosh. Anybody who has Mac OS10 can freely download that and install it.

We’re using Microsoft’s Visual Studio Version 2005 on Windows, and we have an arrangement with Microsoft where they have distributed the software free of charge to students, so if you would like to install that on your own Windows computer, we’ll give you some handouts on Friday that tell you what to do to get the compiler and get it installed on your computer.

Our cluster computers in the dorms and the libraries and the lair have both the compilers and libraries installed, so if you work in a cluster, you don't have to do anything special. You just walk up and it's got the stuff ready to go. I'm a Mac person. I've been a Mac person forever. I can't get over the fact that you go to the start menu to shut the machine down. That makes Windows impossible for me to use. I would say campus wide, there are more Windows machines than Macs on campus, so if you want to take a popular vote, you could do that. If you want to be on the side of the Mac bigot, you can come and be with me in X Code.

I've got ten minutes to tell you a little bit about C++. That is the next journey that we're going to go on together. The first question is why are we doing this to you? I just got comfortable with Java and now you're telling me to throw away my Java and start over. Let's generate a little bit of love for C++. The advantages of early multilingualism – I have two small children at home that are two and four, and I read a lot about bilingualism. It's very clear that for natural languages, when you're acquiring a language at those young ages that that is the best time to introduce a second or third language.

It's been looked at in terms of programming languages as well, that when you are learning a programming language, there are certain kind of ruts your mind gets into about the way a language is that is based on your early experiences. If you spend a very long time working only in one language, those ruts get deeper, and you have a certain way of thinking. You're a little bit stuck in that paradigm and its approach. What's easy to do in that language, what's hard to do in that language tends to make a stronger impression on you in a way that makes it harder as you grow and explore the languages to kind of get out of those ruts and shake yourself out of it.

There's been some pretty good evidence that somewhere between one and two is a good time to think about branching out and starting to think about different ways of doing stuff and seeing some different syntax and some different ideas to help build in the flexibility from an early age in your career to buy you some strength later. That's part of what we're doing.

Another part of it is actually that a lot of our upper division courses rely on a knowledge of C and C++, that family of languages, and that the longer we postpone it, the more painful it becomes. In the later courses where you're learning about compilers or graphics or networking, they don't have the time in those classes to stop and teach you C or C++. They need you to know C++ to get the work done. Moving the foundation into a programming class seems to make the most sense in the context of our curriculum.

We do switch you over here. The good news is that it's not as big a change as it might sound at first glance. Java is actually highly derivative of C++ for a start. They're kind of cousins in the scheme of language design. They have a lot of syntax. How much C++ do you need to know to start? The answer is zero. You don't even need to know what the word means. In fact, you actually probably know a surprising amount about C++ just by virtue of what you already know just translated a little bit.

Things like the four loop of Java or the way you declare variables or the way parameters are passed into a function is exhibited in Java in very much the same way it is in C++. There are a bunch of things you already know and you don't even realize. It is not assumed that you know C or C++. If you happen to already know those things, you're ahead of the game. If you have not, then not to worry.

How much C++ are you going to learn? We will spend the first three or four lectures just talking about how things get expressed differently in C++ and mostly talking about the differences in the libraries. The syntax itself is quite similar. Some of the more extensive changes have to do with how the C++ string is operated on versus the Java string.

How you do file input and output reading in C++ is a little bit different than the way it's done in Java. We'll spend some time saying here are some things you know how to express in one language. We're going to teach you how to express them in another language. It's just mapping from your previous vocabulary onto a new one.

Along the way, we will actually introduce some of the C++ features that we need to support our pedagogical goals. We'll be talking about how classes get designed in C++. We'll see how to use classes. We'll see how to define those classes, and we'll look at things like templates, which is the C++ construct for doing generics that you seen in Java – how you can build containers that are type unspecific and things like that.

We will learn a little bit of some of the fancier features of C++ like the pass by reference parameter, but there's actually a very large amount of C++ that's just off the table for us. We will not make extensive use of the standard template library or the fancier features of static and [inaudible] and a bunch of key words that mean nothing to you and should mean nothing to you. You will learn enough to have reading familiarity with C++ and to be able to express yourself quite well in the subset we're using, but it is a subset of C++ that you're being exposed to.

If you find yourself really wanting to master C++, we are offering CS106L. CS106L is a lab companion course that is open to students enrolled in 106B or 106X. It meets twice a week. It's actually on Monday and Wednesday late afternoon. It's 4:15 in Hewlett 103.

It's being taught by a veteran section leader who is very well versed in C++ and who of his own volition volunteered and created this course because he himself was a little frustrated as a prior 106 student in wanting to get at some of those C++ things that weren't fitting with our goals. It is a place where you can get more exposure to standard C++, do some exercises that help you to test out those things and see how those things are expressed and get another unit. It's a pass/fail lab course.

You can also just attend or grab the materials if you just want to look at them. It's a great way to broaden that knowledge of C++ beyond what's useful for us in terms of our goals. I'm going to ask you some questions, because I don't get to do all the talking. How much C++ do you need to know? Some of you may know nothing about it, and that is perfectly fine. Some of you probably know something about it or at least have heard something about it. I'm going to have you guys tell me what it is that people tell you about C++ that makes you either excited to learn it or frightened to learn it or interested in how it works.

Student: It's what Java is based off of.

Instructor (Julie Zelenski): It is what Java is based off of. That should be a little reassuring that there is a syntax there that got adopted with some minor changes. It should feel more familiar than different when you look at it. It was very strongly influenced by a generation of programmers who C++ was their native language who designed the Java language. That's a good thing to know. What else do you know about C++?

Student: It's extended C.

Instructor (Julie Zelenski): Yes. It's extended C. Here's how it fits in the spectrum. C is kind of a 1970s creation. C++ is a decade later. C is the language which it is based on. It is an extended C. It's called a superset. Everything that compiles and works in C still exists in C++, but then they added a bunch of features. Not only did they add a bunch of features but they tried to fix some of the things about C++ by replacing existing things.

For example, there's some string handling in C that's kind of very primitive. They added a string object with much cleaner handling and safer semantics into the C++, but they kind of left the old one around. Some parts of C++ feel a little strange because of this history to it – the legacy of incorporating everything C was plus the stuff means at times there's a little bit of weirdness there. It also means that the language, as a result, is very large. C's safety and runtime features were extended by what got added in C++.

Student: A friend of mine told me that [inaudible].

Instructor (Julie Zelenski): That's a good thing to know. C++ might be a little bit more dangerous than Java. That is true. Java is very concerned with safety, in particular since Java was designed for web delivery of content. It was very important that the program have very constrained features on what it can and can't do, and so as a result, Java tends to be very parental. When you forget to initialize a variable or forget to return from a function, Java's very aggressive about saying hey, you've got to fix this.

C++ is a little less parental. Here's the overly protective mom, helicopter mom, the ones who stand. That's Java. Java's making sure – oh, are you okay, sweetie? Let me stand here in case you fall. C++ is crack mom. She's like yeah, I'm over here with my friends. Don't play with the kitchen knives. It's a professional's tool, and professionals don't want to feel encumbered. There are certain things they want to do that require some of this low-level access, and safety usually comes at a cost.

Any sort of feature where the language is double-checking for you is taking time and efficiency. There's a cost associated with that. Every time you want to get something out of an array, it's checked to make sure that that number was not off either end. Every array access costs you a little bit more. C++ says I'm not going to charge everybody that penalty. If you actually have the bad sense to write a program that does those things, you deserve to be punished. As a result, you will at some point in this quarter get to experience some of that firsthand.

Some of it is a growth experience. Some of it can be frustrating. It is part of what professional tools often look like. They are making these tradeoffs of efficiency over safety that put more of the work back on you as the programmer to be a little more attentive on those things. You can write programs that crash in very spectacular ways much more interesting and varied and dangerous than the kind of things you can do in Java. Good to know.

Instructor (Julie Zelenski): You want to use pointers. You're going to get to use pointers. Pointers are really neat, and they're also very challenging. Pointers are these ways of building these very flexible and amazing data structures – the kind of things that we're going to try to build. At some point, they are going to be the only way to achieve those things well, so building these things called trees, graphs and lifts rely on understanding a mastery of the pointer type.

The pointer type is complicated, and it's part of that danger thing, which is having access to rearranging memory by virtue of addresses opens up a lot of opportunity for there to be mistakes. Mistakes can be made in the passive voice that have consequences. You'll get to experience firsthand what that's like. There's joy in it, because getting it right is awesome, and there are things that you can achieve that are really extraordinary with pointers, but when it's not working, it can be frustrating. You're getting a little bit of both.

It turns out C++ does not have a graphics system built into it. Java is actually distinguished from previous languages. Java tries to solve all problems. Traditionally, a programming language tends to have a set of libraries that have facilities for data management, reading and writing files and sometimes some networking, but they don't tend to actually solve application layer problems. That tended to be a different piece of technology. The Mac OS might offer a graphics library that was written in C++, but C++ the language didn't have a windowing system or graphics system.

C++ does not have those features itself, so any C++ compiler you get comes with these basic things about handling files and managing these types of data structures, but it does not come a priori with a bunch of graphics routines. That said, Windows and Mac and Linux and all these things have graphics routines that are written in C++, but they're all different. To say what are they like relative to Java? They're all different relative to Java. It's not standardized.

There's a wide variety of them out there, and they tend to have a lot of very impressive and different solved problems, because C++ has a longer history than Java that a lot of problems have been solved in C++ that are available to you, too. There's a lot of existing code other than what you might think is standard. I will see you on Friday, and we’ll be seeing some C++. If you have questions about your situation, come and talk to me now.

[End of Audio]

Duration: 43 minutes