ProgrammingAbstractions-Lecture02

Instructor (Julie Zelenski):Welcome. A couple things that I wanted to go through administratively to get us all up to speed – if you did miss Wednesday's class, the handouts are available on the web and the online video also, so you can take a look at what we did. We just did a course overview and a little marketing and just gave you some ideas of what we're planning to do together this quarter to make sure you're in the right place. That might be a good thing to review if you did happen to miss is.

The section times are now available on the web. They went up last night. They'll be up for the next couple of days. You go in and list the preferences of times that fit into your schedule, and then we do this big matching to get everybody into a section that works. There's no reason to do it early versus later. We do all the matching once we have everybody's preferences in. You do want to do your preferences at some point before it closes so we can make sure we have your information in the mix.

If you forget to do this, you'll be at the mercy of where we can fit you, and that might be much less convenient for you or, in fact, even impossible. Be sure to get your preferences in for us to do the match with. One thing I should have spent a little bit of time on on Wednesday – how the honor code applies in computer science and how to make sure that all of us are doing our part to uphold the honorable community that we have pledged to be a part of.

It's a little bit unfortunate and a little bit of an embarrassment to computer scientists that we actually account for a disproportionately large number of cases that are brought before the judicial board at Stanford. My personal thought about that is not that we somehow attract the dishonest student, but I think there are certain things that combine to make some of the pressures of CS a little bit outstanding relative to some of the other things. If you're up working late and you're trying to write a paper and it's not really coming together, at some point, you can say it's good enough and you can send it off, even if it is not your best work.

It's a lot harder to do that when you have a program that doesn't do what it's supposed to do. It may crash right away or give the blue screen of death, and that feeling of failure or lack of accomplishment is one that's hard to stomach as the talented people you are. Sometimes, that leads you into temptation about how you might be able to get the program working through means that aren't really your own work.

Let me be clear about what our rules are. I put out a big handout last time. There's also a big follow-up thing on the web that I think is worth reading, because I think everybody needs to know where the boundaries are. In general, you need to be doing your own work. Your own work means you're sitting down and coding and thinking and designing and debugging independently. That said, we have a lot of staff hours that can help you when you get into trouble, and asking questions about any general features of the language or the library or how this works or strategies is totally fine.

You should definitely rely on each other to get through those rough spots. When it gets to the point where you're looking at somebody else's code or looking at code together or looking at someone's code from a passporter or giving your code to someone else, you're crossing that boundary we don't want you to cross. When that happens, it often leads to the code coming out much more similar than you might imagine. In fact, we use automatic tools to help us identify situations where that might come up. We do move forward with bringing those cases to judicial.

It's personally a very sad thing for me. I want to get this out now. The point is we're here to help you, and we want you to succeed. If you're getting stuck and you're having trouble, the right thing to do is to ask us for help. We can often get you past that sticking point and get you onto doing the right work for yourself, but using someone else's code – people do make big efforts to try to conceal what they've done, and it doesn't work is the truth. We do catch it. The result is not a pretty outcome for anyone.

That said, you do not need to worry about us mistakenly identifying your code as looking like someone else's because we're asking you to solve the same problems. It turns out that code is as individual as a fingerprint. The way two people are writing an essay about the treatment of class in Dickens' "A Christmas Carol" don't look the same. It doesn't matter that we ask you to do the same thing. They really do look different. You have nothing to worry about as long as you are doing your own work.

Please do read those things, the handout and the web stuff just to go through all the minutia of how to know when you're getting into those gray areas, and certainly ask if you have any questions. About the alternate final exam – I don't want to get into the slippery slope of having much alternate exams. That actually creates issues with security and fairness that I'm not prepared to figure out how to navigate, but I did get a certain number of requests, and I've decided to be a little bit more lenient than I usually am.

I'm going to offer exactly one alternate final time, which is 24 hours earlier than our regular exam. We'll have more information about that as we get close to that, but for people who have an exact conflict or need to leave a day early, that will accommodate you. I will make no other accommodations. If this means you cannot take this class, then don't take this class. I am not capable of managing an excessive number of people who all want to take it at a time that works best for them. The university's scheduled slot is Friday, 12:15 to 3:15. Everyone really should have that time available, but if that for some reason is completely unavoidable for you, then Thursday, 12:15 is it.

One thing I'm going to try to do on Fridays is take a little time to sit and talk about stuff and having something less formal than office hours. It's not really about having CS questions answered. It's just getting to know each other and having a chance to talk. I think most students really wish they had spent more time with their professors, and I think I'm a fascinating and interesting person. That's something that comes up again and again is the feeling that, especially in a large class like this, the feeling of being anonymous and not being able to make that connection. It's something that I'd like to try to fix, and I want you to try to fix it with me.

Fridays, there's a school of engineering undergraduate council meeting that will meet every third Friday, but on the other two, my plan is to walk out of here and go over to that café that's down in the basement of Terman and get a nice cup of coffee and a cookie and sit down and talk. You're welcome to come and talk about anything and everything that's on your mind and just get to know me. If not this week, then maybe some other Friday will work for you. I will be happy to make your acquaintance.

What I'm going to talk about today – I'm going to do some C++. I'm going to talk about how C++ relates to Java, show you some sample programs, do a little coding with you, see if you can kind of get the feel for how things are going to work, try to talk about some of the specific mechanisms that are different in C++ that we need to get familiar with. The reading that goes along with this is the handout four, which tries to go through the high-level overview of the Java/C++ differences as well as the reader chapters, which fill out more of the details about the language.

What we're moving onto next week is the C++ library, so we're talking about string and stream and the CS106 libraries that go along with it that form the foundation of things we have to build on. Any questions administratively? Anyone read the handout two and had some questions about policies, procedures, or stuff I talked about Wednesday?

I also brought something. A former student of mine brought me her homemade caramels, and if they sit in my office, I'm going to eat them all, and I don't want to do that, even though they are extremely delicious, and I thought I would bring them as a bribery offering where it's hard to talk in a class that's this big, and you feel like everybody's staring at you, but I figure I'm going to offer up sugar as reward for courage, and so I'll have this little bag of stuff here, and if you have a question, just figure not only will you get your question answered magnificently, there will be a bonus butter/sugar bolus that goes with it.

This is Java. It's like a death match. What's the same? The first thing I want to talk about is that there are a lot of things that are the same, and so you are building on a background that is going to serve you well without having to make a lot of adjustments. They're almost so similar that at times, it can be a little bit unhelpful in the sense that it's easy to get tripped up because they almost but not quite are exactly the same. There was one unfortunate quarter where I was teaching two classes – one that was being taught in C++ and one that was in Java, and I was a mess the whole quarter.

Luckily, I'm not doing that now, so hopefully, I'll have my C++ hat on straight the whole quarter. The general syntax – I'm going to mention these things, and then I'm going to show you a sample program. Things like the comment sequence, the slash star or the slash slash – the use of almost all the punctuation characters is exactly the same. The statement terminator that the semicolon is used for, the commas to separate arguments going into a function call, the way you make variable and parameter declarations – exactly the same.

All of the primitive variable types – char, int, double, plus the kind of more esoteric log and float and short and things like that that are present in Java are present in C++ and have basically the same semantics. There is one minor adjustment there, which is Java's Boolean type, the true/false variable type is actually called Bool in C++, so you have to type a few less characters on that end.

All of the operators that manipulate the basic types, the assignment operator [inaudible] the equal sign, the equals equals to compare, not equals to see if they're not equal, the arithmetic plus, the plus equal, multiply equal shorthands, the plus plus that comes in both pre and post increment, the relational less than, less than or equal to, the logical operators and/or have the same behaviors. They have the same short circuiting, the same precedence table, so all of that is without change.

The control structure – so for redirecting control flow and making choices about what code to execute when, the floor and [inaudible] mechanisms, the if with the optional else, the switch statement, which you may or may not have seen, but you can definitely look up in the text if you haven't – that kind of has a variant of the if else, so it lets you decide based on the value of something which case to move into, the return that's used for the function – also things like the break and continue, a little less known keywords there or the do while – they work the same way in Java and C++.

Let's look at a complete C++ program. This one is printed in handout four, if you want to follow along there and make notes on it as we talk you through. What I'm going to do here is I'm actually planning on just walking through this line by line to kind of point out some of the things that are different. Then I'm going to move to the compiler, and we'll do a little bit of experimentation with developing and writing code and manipulating something that looks like this to get some of our bearings about what things are going on.

First off, just some general structure. A C++ program is traditionally written in a text file with a suffix some name.CPP. There's actually some other names that get used like .CC or .C that are also accepted by a lot of compilers. Some text extension that identifies that this file has C++ code – we'll use CPP. The program is traditionally in a simple form just one file with all of the code that's necessary to be in there.

Let's take a look at it. This one starts with a comment – always a good idea. This is average.CPP, and it adds scores and prints their average. It's just like all the good style habits that you have in Java. The next three lines are #include statements. #include is analogous to the Java import. The C++ compiler is pretty rigid about wanting to see things in the proper order. Before you start using some facility, it wants to know about it. That's part of the safety of making sure you're using it correctly.

If you plan on drawing something in the screen and doing some graphics, it wants to be sure you're making calls to the graphics function correctly with the right arguments, the right names, the right arrangements, and so in order to know that, it has to ahead of time have seen that bit of structure. The way this is done in C++ is through these interface or

include files. The #include mechanism is fairly primitive. It basically says look for a file with the name genlib.h and take its contents and dump them right here.

The contents of genlib.h and simpio.h are actually to give you a little insight. We're going to see them more later. It lists the operations and facilities that are available in this particular header that we're trying to include. In this case, the genlib is a 106 specific header that all our programs will include. It gets us set up with some basic foundations. The simpio is our simplified IO header that adds some features for interacting with the user at the console.

The next one down there is include . is a C++ standard header, and that helps to point out the difference between why in one place I was using double quotes to name the header file I'm looking for and in one place I'm using angle brackets. This is a way that's distinguished for what header files are, what are called system standard, and it looks for them in a particular place where the standard header files are. Those are always enclosed in the angle brackets. In double quotes are local headers that are specific to this project or this environment that are not C++ standard.

Whenever you see these double quotes, you'll think it's my own local headers or headers from the CS106 library. Anything in angle braces is a system standard header you'd find everywhere. It matches what you do in import. You say I'm going to be using these features in this class. Let me import its features into here so that the compiler knows about them.

The next line I've got here is a declaration of a constant. This is the const whose name is NumScores, and so it looks a little bit like a variable declaration. From this point over, you see I have int NumScores = 4;. It looks a lot like a variable operation. The const in the front of it is our way of telling the complier that this particular value once assigned will not change. It is equivalent to kind of the use of final in the Java language in many ways.

This is being declared outside of context. One point to make here is that C++ drives on more than one way of expressing code and what we call its paradigm. Java is the completely object oriented language. Every line of code you wrote went inside a class. It's all about classes. You wrote this class that manipulated the thermometer. You wrote this class that drew with the turtle. There was no code that ever existed outside of a class. You started off with public class something and you started writing methods that operated on that class. Maybe they were static or maybe not. The way you started code was always to say start from this class' main to get work done.

C++ is a hybrid language. It has object oriented features. It has the ability to find and use classes, and we'll see that quite extensively, but it also inherits the legacy of C, which is pre this whole object oriented development where it operates more procedurally, they call it, where there are functions that exist at the global level that don't have a context of a class that they operate in. There's not some [inaudible] that's being received. They're just functions that exist outside. It's a little set of things to do that don't live inside a class.

For much of the term, we're going to be doing stuff in a very procedural way, at least from the code we write, and we will use a lot of objects. We'll actually be mixing and matching a little bit of those paradigms together, so using the procedural paradigm to write a lot of code that happens to manipulate objects using an object oriented set of features. In this case, the program here – this is actually up at the top level, and it's saying this is a global constant that's available anywhere in the file. After this declaration, I can use the NumScores, which will refer back to this without any further adornment.

The next thing underneath that is a prototype. This is actually a little bit of a novelty that in Java you are allowed to declare if you have three methods, A, B and C, and maybe A calls B and B calls C, you could declare A, B and C in any order that's convenient for you. You want to put them alphabetically? You want to put them in the order they're called? You want to put them in the order top to bottom down or bottom up? It doesn't matter. C++ tends to want to look at a file once top to bottom and not have any reason to go back and revisit things.

It happens that in the main function, which is the one where code starts from, I want to make a call to this GetScoresAndAverage function. If I had left this prototype off and I had this code here, when it gets to this line where I'm making that call, the compiler hasn't yet seen anything about GetScoresAndAverage. It's on the next page. It will complain. It will say I don't know what the function is. You haven't told me about it. It came out of nowhere.

This prototype is our way of informing the compiler about something that's coming up later, and so the prototype tells about the return type, the name of the function, the type and names of the arguments and the order of them and then ends with a semicolon. It matches exactly the function header that we'll see on the next page, and it just tells the compiler in advance here's something that's coming up later in the file. It's a function with this name and this signature. It's something you didn't have to do in Java but is a part of the C++ world.

There's actually an alternate way I could have done this, which is when I show you the code for GetScoresAndAverage, if I actually defined them in the other order, I could avoid that separate prototype. It's a little bit of a matter of personal taste which way you feel comfortable doing it.

Let's look at this guy. Main is special. Lowercase m-a-i-n has no arguments and returns an integer type. This is the function in the C++ program where execution starts. There can be exactly one main function with this name and this signature, and when the program gets up and running, the first thing it does is start executing the lines of code that come in this function. No objects are created or any kind of fancy outer structure the way it might be in Java. It immediately goes here and starts doing what you told it to do. Main has a special role to play in any program.

What this program does is first prints something. This is the C++ syntax for printing things out. Even if you haven't ever seen it, you probably can get a little bit of an idea of what it's trying to do if you just step back from it and squint a little. It looks a little like system.out.print lin. This is what's called the consult out, or the standard out that's here on the left-hand side. The use of the double less than here is what's called the stream insertion operator. The standard output operation is saying insert into that stream and then in the double quotes is a string.

This program averages. It says insert next to that the NumScores, which is this constant. It got the value from above. And then another string, and at the very end, there's this little endl, which is the end line insertion of what's called a stream manipulator. It says at the end of this, put a new line and start any subsequent text on the next line. In Java, that would look like a system.out.print lin of this program averages plus the NumScores plus that using the string [inaudible] and conversion stuff in Java.

It looks a little bit different in C++, but in the end result, it's a matter of printing out a bunch of numbers and strings intermingled with new lines. Nothing too fancy there. The next line is making that call to that function that we earlier told it about but we haven't yet seen the code for that makes a call to get scores and averages. It passes the constant NumScores and says that's how many scores that parameter has used to decide how many iterations to run the loop requesting the values. Then it returns the average of the values entered and then we take that number.

This just shows that in C++, you can declare variables anywhere, either at the start of a block, in the middle of a block or wherever they are at. We took that value, stored it in this variable and then used it in the next print statement. The last thing we have is a return zero. The signature for main in C++ always returns an integer. That integer is not all that useful in most situations. It tended to be a kind of return that tells the operating system who invoked the program whether everything completed successfully or not.

By default, the value that's used to show successful completion is zero, so unless you have some other reason, it is return zero that you'll always see at the end there. In reality, if you were to return something else, you won't see a lot of other differences. If you return -1 or 642, in most contexts, that number's almost ignored. It is there just to make tidy.

I'm about to flip to the next part, and this is the decomposed function that main made a call out to that does the averaging of the scores. We've got a comment on the top that tells a little bit about how it works. It's always a good idea to get some documentation in there that's helping the reader get oriented. That same matching of the prototype that was given earlier – same name, same argument, same things, but instead of ending with that semicolon, it goes into the body.

Sometimes we'll call these very similar terms – this one we will call the definition or the implementation of a function, and that earlier just introduction of it is called its declaration or its prototype. They should match, and they're just there to give warning of

something coming up later in the file. It initializes a sum to zero. It runs a standard four loop here. Like Java, we tend to count from zero as computer scientists for a long legacy reason. Instead of going from one to NumScores, we actually go from zero to NumScores minus one.

Each time through the loop, we are prompting the user to give us a score. We are reading their score with a GetInteger call, and we'll see what GetInteger is. GetInteger is a CS106 function that retrieves a number that the user types in the console. We add that into the sum and keep going, and after we've done the whole thing, at the very end, we're doing a division of the sum by the number of scores to compute the average score that was [inaudible] and return that.

Student: Did you say why we didn't have end line?

Instructor (Julie Zelenski):In this case, I didn't have an end line just because I think it makes it nicer for the user. If you say next score and end line, then you're typing on the line underneath it, and so if you don't put the end line there, it's like you type right next to the question mark, and so it actually it an aesthetic thing. If I put it there, it would say next score and then I would be typing underneath it, and instead, I'm typing right next to it. I hit the return and then the next one comes on the next line.

So [inaudible] – and I will talk about that a little bit later today, but it is – that's a good question but a little bit advanced, so just take my answer and then we'll get to that in maybe 20 minutes.

Student: So for every main function, you always have to end with return zero?

Instructor (Julie Zelenski):Pretty much. As I said, you could put return anything, but as good form, return zero is a way of saying I successfully completed my job and I'm done.

Student: Do you use a void return type for the main?

Instructor (Julie Zelenski):In C++, no. In C, it's a little more lenient about that, and it will allow that. In C++, [inaudible]. I have to decide if I can get this to you without hurting you.

Let me do a little coding with you guys just to play around with this and show you the complier. I'd like to do a little bit of this. I think it's really easy for me. I talk really fast. You might have noticed. When I get going on something, it's easier for me to put up a lot of code on a slide and say here it is. It's all fully formed. The reality is when you're running your programs yourself, it's not going to come to you fully formed on a slide all ready to go. You're going to have to figure out how do you stack through making this stuff work?

I'm going to do a little bit of development of a program that looks very similar to the one I just wrote, but I'm going to play around with it a little bit to get at some of the things

that are different about C++ and how the tools work. This is basically what an empty project starts me off with. I've got the include for the genlib, but I'll always have my int main and return zero and nothing else doing. I'm going to sit down. I'm going to start typing.

I'm going to say welcome. I'm going to move to – let's do this. We'll play with this for a second. There are some things that I can do. I can say welcome a lot of times. That wasn't really so hot there. You guys can help me when I fail to type correctly. I say welcome a lot of times. Let's go see and make sure this works. Look at that. It doesn't like what I did. Let's see what it has to say.

It says cout was not [inaudible] the scope. Endl was not declared in the scope. This is an error message you're going to see a lot of times in various other symbol lanes that will show up there, which is the C++ compiler's way of saying to you I don't read your mind. You used cout. You used endl. These things come from somewhere, but they're not known to me until you make them known to me. The way I make those known is that those are part of the standard IO stream, which the input/output streams facility for the C++ language, and once I tell it about those things, it's going to be much happier, and it's going to say welcome to me a lot of times.

In fact, I could say this – 106B rocks. I'll do this. I'll say how awesome equals get integer and I will – it may help if I move this up a little bit so we can see how much do you love 106B? Here, I won't use my endl, so we'll see how the prompt comes out. I'm going to get an integer from them, and I'm going to use that to write how awesome that many times. What does it not like about that?

Again, it says get integer not declared in the scope. This is going to be the bane of our existence today, which is remembering that there are headers out there that have features that we're going to be using, and getting familiar with the ones that we're gonna need – there are probably about ten total that we'll use again and again, and we have to get our bearings about which ones are where. The documentation for all of our CS106 headers is available on the website in a nice HTML-ized version, and the ones from the standard libraries are detailed in the book in chapter three.

How much do you love 106B? I could say well, I love it two times. I say no, I love it 1,900 and so. 10,000 times later, if you're not convinced now, I don't know what will convince you. We're seeing some stuff. Let's see if we can make that averaging thing work for us. I'll go back to having it say welcome. That's what the endl was doing for me there. The prompt is waiting at the end of that line. If I have the endl, it will actually be waiting on the next line.

Let me put a bogus message here. Then I'm going to say double average equals GetScoresAndAverage and then let's go ahead and – I'm going to blow off the constant for a little bit because I'm going to be mucking with it anyway, and I'm not going to use that constant for very long, so I'm not going to worry about it yet.

I put my prototype up here so it knows what I'm talking about, and then I'm going to print out average is. I've got a little bit of structure there. I can even go in here. Copy and paste is a really good idea. Once I've written the function header once, I don't really want to make a mistake. Something that's interesting to know is that you can always write code that doesn't do what you want to start with but that lets you test some of the other pieces around it.

In this case, I haven't finished fleshing out what GetScoresAndAverage is, but I can write code around it that says well, it passes in this number and it does some stuff. Right now, it just returns zero. It's totally bogus. It's not solving any of my problems, but it does allow me to test some other part of the code to make sure that connection and other parts are working before I go on to work on this part in isolation.

One of the things I'll try to show you whenever I'm doing code is to try to give you a little of the insight to how I approach thinking about the code. I do think that one of the things that really distinguishes somebody who works effectively and productively from someone who spends a lot more time doing it is often just strategies and tactics for how you attack your problem. I'm hoping that along the way, I can kind of share a little bit of the insights that I use to keep myself making forward progress.

I am going to put in that four loop that I'm looking for. I'm really not this bad of a typist in real life, but it's easy to – the use of the braces here is what's defining the block structure. Without the open curly brace close curly brace, the four loop and the if and the other statements only grab that next line as being part of the indented body of the four loop.

I'm going to go ahead with my next idea. I'm going to put that in the sum, and then I'm going to do some things that seem bogus. We're going to get there. I'm going to make some mistakes. Some of them I'll make on purpose. Some of them I'll actually be making accidentally. We'll see which ones I'm better at finding.

I have my GetScoresAndAverage. It asks them for the scores. It adds it into the integer and then doesn't return anything. First, let's see how the compiler feels about that. The first thing you'll notice is that compiled. That might alarm you just a little bit, because if you go back and look at this code, it certainly seems wrong. Anybody want to help me with one of the things that's wrong with it? It doesn't return anything. That seems a little shocking. It doesn't return anything at all. What is it then actually going to produce?

What is it doing that is a little bit bogus? Sum is not initialized. These are two examples of errors that Java doesn't let you get away with. You may very well remember from having typed in some code like this in Java that it will complain. If you have a variable and you declare it and you don't initialize it and you start using it, the compiler says not happening. You need to go back there and give that guy an initial value before you start reading it.

Similarly, you fall off the end of a function. It said it was going to return a double – you better return a double. Every code path needs to return a double. You can't bail out early in some cases and leave it fall off the end. It considers both of those big enough errors that it won't let you get away with them. C++ compilers are a little bit more lax. It's crack mom, I told you. It's like well, if you don't want to initialize that, go ahead. What value does it have then?

It doesn't make it zero. It doesn't make it empty. It doesn't do anything nice or clever or helpful. It's like whatever kind of contents were lying around in that particular location, now it's [inaudible] initial value. The result will be that we could get some pretty strange results. If I run this thing right now – let's see. I type in some numbers. I type in somebody got a nine, somebody got a 67, somebody got an 87. I type in a bunch of numbers, and the average is zero. That was interesting. Zero seems like it had some rhyme or reason to it, but there's no guarantee. I could run it again and I might get very different numbers.

I got zero the second time. Good luck for me. Let's go in and start fixing some of this stuff. The thing to note is the compiler isn't nearly as aggressive about being on your case about things, so it does require a little bit more self-monitoring on some of these things. Seeing this result and going back and figuring out how I got into this situation – let me get that guy an initial value, and I'll go ahead and put a return down here.

We'll see if I like how this works any better. I put in five, six, seven and six, and the average is six. It kind of looks good, right? It makes sense to me that the five and seven cancelled each other out. But if I do a little bit more deep testing where I put in numbers like five, six, five and six, my average is five. In this case, I would think that I would be getting something like five and a half.

I go back here and take a look at what I did. I took sum and divided it by NumScores. What have I failed to do? Cast that thing to get it out of there. The default – this is the same in Java as it is in C++ is that if you are combining two operands in an arithmetic expression, if they are of mixed type – one is double and one is int – it promotes to the richer type. If one of those was a double, it would convert the other one to a double and do the calculation in double space. As it is, sum is an integer and NumScores is an integer. It's doing integer division. Integer division has [inaudible] built into it, so if I divide ten by three, I get three and that remaining one third is just thrown away.

Even though the result is double, that happens after the fact. It does the calculation integer space – the ten divided by three. It gets the three-integer result and then it says oh, I have an integer here. I need to convert it to a double on my way out. It takes that and turns it into 3.0. It doesn't recover that truncated part that got thrown away. If I really want to get a real double result, I've got to make one of those a double to force the calculation into the double space.

The mechanism of C++ is a little bit different than it is in Java. In fact, the Java mechanism does work, but the preferred form in C++ looks a little bit like this where you

take the name of the type you're trying to convert it to. It looks a little bit like you're making a function call-passing sum to a function that will change it into a double. By doing this, I now have one of them being double. Int on the other side gets promoted. If I do my five, six, five, six, that truncated part is now part of that calculation and preserved and pulled out and printed.

Let me change this a little bit to something else. Right now, it assumes that there's a fixed number of scores that you want to do. Maybe what I have is a big stack of exam papers, and I don't actually know how many are in the pile, and I don't actually want to count them ahead of time. I just want to keep entering scores until I get to the bottom of the pile. I'm going to change this loop to allow me to enter numbers until I say I'm done. Let's take a look at what that's gonna look like.

I'm going to get the value here. I'm going to change my code a little bit. Then if the value is the sentinel, then I don't want to include it in the sum. Otherwise, I do. Maybe we'll do this. We'll say if the value doesn't equal the sentinel, then sum adds into there. I also need to make this thing stop, and so I'm going to have to rearrange my loop a little bit here, because if you think about what I'm going to try to do, I'm going to prompt and get a value and then either add it in the sum or quit.

Then I'm going to prompt and get a value and add it in the sum or quit. I have a loop where I need to rearrange my notion a little bit here. One way I could do this is something like this. I could say this. While value does not equal sentinel, then add it into the sum. I'm going to kind of change my code around. Watch carefully as I reorganize it.

If the value is not equal to the sentinel, then I'm going to add it into the sum and then I'm going to get the next and overwrite the previous value. I inverted that loop that was there a little bit. It says while the value is not the sentinel, add it into the sum and then get the next value and come back around.

The problem with this code as it stands is there's something a little bit wrong about value. It's not initialized. I need to initialize it to something. What do I need to initialize it to? What the user wants. This little piece of code down here needs to come up and be repeated up here. I need to get a value from them and then go into if it wasn't a sentinel add it and get the next one.

This is what's called a classic loop and a half construction where there's a few things I need to do and then I need to do some tests and decide whether to go on with that iteration. As it is, there's half of the loop that starts up. They call it priming the loop. That's a little bit unseemly. There's something about that that's a little bothersome. Even though it's a very small piece of code – two lines of code is not going to rock the world.

I do want to try to get us into the habit of thinking if we can combine that code and unify it, that's better. It's better than repeating it. Repeating it means that if I ever change something about it, I have to change it in two places. There's an opportunity for error to creep in that I can avoid if I can get it to where there's really just one and only one

version of the truth. I'm going to change this. It's going to use the break statement, which you may have had a little experience with.

I'm going to run a while true loop, which looks like it's going to run forever. I'm going to prompt to get the value and if the value equals the sentinel, then I'm going to use break and immediately exit the loop here without completing the bottom of this loop iteration. It causes it to move on to the statements after the loop. By doing it this way, I now have one prompt, then a check, and based on that check, I either finish this iteration and keep coming around or I immediately break out saying that was the clue that we're done with this.

I would say that it's a little bit of an advanced question, but it turns out that the do while loop is not really that much cleaner in the end because you still have to do the prompt and test and decide when to add it in. Do well loops are just so unusual that they cause everybody to slow down a little bit when you read them. If you can write it using a more [inaudible] construct, there is some advantage to that.

I got this guy together here, and then maybe I should actually tell the user – this might be a good thing to say. Tell them what the sentinel is. I have to type better than I do. NumScores equals zero, and then each time we get one, we increment. Let's go back. We'll change our call to use a more ordinary value like -1. We'll type in five, six, seven, eight, nine and then -1, and the average of those is seven.

It should seem very familiar in a lot of ways but there are a few details that need to be different. I'm going to show you one of the little C++isms while I'm here because it's something that comes up in the standard libraries and it's worth knowing. There are some minor conveniences in the way that C++ provides certain facilities that are not conceptually a big deal, but you do need to know about them because you're going to encounter them in various interfaces.

One is the notion of a default argument. In the prototype for a function, there are arguments to a function where it's some very large percentage of the time always going to want to be a certain value, but you still want to leave it open for the user to specify a different value in the cases where they don't want to use that standard value.

One way of doing that in C++ is to write your function using a default argument where I say int sentinel and I said equals -1. That is providing for if somebody makes a call to GetScoresAndAverage passing an argument, it's used instead. If they pass nothing, so they just have open paren close paren and they don't give a value for that argument, that means the assumption is to use that default. That means that most people who are making calls to GetScoresAndAverage can just drop this number and get the behavior of using -1 as the sentinel. In the case where for one reason or another -1 was one of the valid values that you could potentially enter, you could pick a different one.

We'll see that in the libraries. It's interesting. The mechanism of it is really quite simple. It's just a convenience to provide for. You can actually have more than one default

argument. The idea is that you can only leave off when you're making the call the last most argument, and then from there, other ones, because it has to figure out how to match them up. It matches the arguments left and right, and as soon as it runs out of arguments, it assumes everything from there has to be used its default argument for it to match that call.

It allows for – you could have three arguments of which you could specify two or one or three if they all had defaults if you needed that. It's not that common. Let me go back to my slides and tell you about a couple other mechanisms of some C++ features that are new to us that we want to know a little bit about. There are two types of user-defined types that are very simple to get your head around. You want to know what the syntax for them is and what they do.

It's the enumerated type or the enumeration where at the top of your program – up there where I was defining constants and doing #includes, this is where we put information for the compiler that is of use across the entire program. This was also where we would put new user defined types. In this case, I want to define a type direction T which can take on one of the four values, north, south, east or west. That whole package up there – that enum direction T north south east west is the way of defining the new type.

You're saying direction T now comes into the space as a name that can be used for variables, parameters, return types – any place you could have used int you can start using direction T. It has the expectation that variables that are declared in direction T will be holding one of those four values. It's like north, south, east and west got defined as constants, and they are, by default, assigned the values zero, one, two and three in order unless you do anything special.

You can use them – you can do things on enums that are very integer like. You can assign them. You can compare them. You could use less than and greater than. You can add them. There are things – they are largely implemented underneath the scenes as numeric types, but they're a nice convenience for making it clear that this thing isn't just any old ordinary integer. It's specifically – it should be kept into this constrained range. It means something a little different. It's just a nicety. It does not have a lot of heavy weight feature associated with it, but it's just a nice way of documenting that something is this kind of set up.

This one – the record of the [inaudible] type – much more broadly useful. It's just an aggregate where you can take a set of fields of same or different type, give them names and aggregate them together. You say here is student record, and the student has a name, dorm room, phone number and transcript. Aggregate it together into a new structure type.

In this case, the point T [inaudible], and so like this – this is the kind of thing you put up at the top of your code that says here's what's in a point T, the field names and their types, and a little point of noteworthy error producer is that there really does have to be a semicolon at the end of this. Similarly, there has to be a semicolon at the end of the enum for direction T.

If you forget to do that, there's a cascade of errors out of that that are really a little bit mystical the first time you see them. You learn to identify this quickly later on. It ends that declaration and then allows the composite to move on to the next thing, not assuming you're still doing more work with this. Once you have this type declared, you can make variables, return types, parameters, all that stuff, and then the access to the members within the fields within a [inaudible] looks like access to an object did in Java where you have the variable name on the left and then a dot and then on the right, the name of the field you're accessing, setting, and reading.

You can do things like P = Q which does a full assignment of one [inaudible] onto another, so it copies all the fields over from one to the other. It's something simple that does have a lot of usefulness. It makes a lot of sense when you have a program and you do group things together. Here's all the information about this part of my data structure – a student, a class, a dorm – all the information being aggregated together into one unit is actually a nice way to keep your data organized.

There are two point T variables. One is P. One is [inaudible]. I'm saying P.X. I'm saying the X field of the P variable is to be zero. At this point, the P.Y's field is nonsense. It's just garbage. I said P = Q, which basically says take the nonsense in Q and override it onto P and make P be as nonsensical as Q is in terms of its contents. Not very useful code.

The last thing I'm going to show you today is to talk a little bit about parameter passing. This is going to come up again and again, but this is just kind of a quick first mention of these things. Someone had asked earlier what is the parameter passing mechanism that's in play? The default parameter passing mechanism is what's called pass by value. It copies. If I have a function here binkie int X and Y, in the body of it, it tries to do something like double the value of binkie and reset Y to zero.

When I make a call to binkie and I had A set to four and B to 20, when I made the call to binkie, the pass by value really means that the X and Y parameters of binkie are copies of A and B. They are distinct. They are new integer variables, new space, new storage, and they got their initial values by taking the current values of A and B and copying them. Changes to X and Y affect binkie's context only. That four that came in here and got doubled to eight, that Y that got set to zero are live here, but when binkie exits and we get back to main, A and B still are four and 20.

It just did full copies of all the variables, and this is true for all types of [inaudible] and enums and ints and chars and all that. It's copying the data and operating on a copy. In most situations, that's actually fairly appropriate. You tend to be passing information in so it can do some manipulations.

In the situation where you really want to be passing in and having changes be permanent to it, there is an alternate declaration where you and an & to the type. Instead of being an int, it's an int&, and that changes this parameter from being a pass by value to a pass by reference or a reference parameter. In such a case, when I make a call to binkie, this first

argument will not be a copy. The second argument's still a copy because I haven't changed anything about it, but when I say binkie of A and B, what the binkie function is actually getting in that first argument is not a copy of the value four.

It's getting a reference back to the original A. For the Y parameter, it's getting a copy of 20. When it makes this change over here of trying to take X and double its value, it really did reach back out and change A. After this, A would be eight. B would still be 20. It allows for you to pass some data in and have it be manipulated and changed, so updated, adjusted and whatnot and then come back out and see those changes that is for certain situations very useful.

This mechanism doesn't exist in Java. There's not a pass by reference mechanism, so this is likely to seem a little bit bizarre at first. We will see this a lot, so this is maybe just our first introduction to this. We're going to come back to this more and more. One thing I will note is that the primary purpose of this is to allow you to kind of change some data. It also gets used in a lot of situations just for efficiency reasons where if you have a large piece of data you're passing in a big database, to copy it could be expensive.

Copying just so you could hand it off to a function to print it would be unnecessary, and so by using the reference parameter in those situations, you're actually just allowing it to avoid that copy overhead and still get access to the data. Sometimes, you'll see it used both for changing as well as efficiency.

I'm not going to show my last slide, but that's what we're going to be talking about on Monday. We'll talk about libraries. Looking at chapter three is the place to go in the reader. I'm going to be walking over to Terman momentarily, and I would love to have somebody come and hang out with me. Otherwise, have a good weekend. You have nothing to do for me. Enjoy it. It's the last weekend in which you will not be my slave.

[End of Audio]

Duration: 52 minutes