Week 4 Module 8 Functions Transcript

k-Combinations Problem 1.2

Hi there hope you are having a great day. Today we are going to talk about functions. Let's start by first calculating the k combinations. Now let's start with the definition what are k combinations. So let n and k be two integers non negative integers. And let's assume that k is not greater than n namely k is less or equal to n. So the k combinations are the number of unsorted selections of distinct elements from a set of elements. So assuming you have n elements in how many ways can you take k elements out of that. For example if n is five and k is three what are three combinations that case. So assuming you have a set of n elements in this case five balls a green one orange purple blue and red. Let's see in how many ways can we select three balls out of these five balls. So the first one can be I don't know take a blue purple and a green. So they aren't ordered it means that there is no difference between blue purple and green and green purple and blue. It is the same it is choosing these three balls. And that's another way. And there are no duplicates here. So we can't choose two purples or something like that. So we can do it this way that way we can do it in a lot of ways. The number of ways we can choose three balls out of these n out of these five are called three combinations. So we would like to figure out what are three combinations from a set of five. Luckily for us by the way it is also noted as n choose three that's another way to call it. And we can write it as like this big parentheses and then you have a n and a k under it. Luckily for us there is a theorem that tells us the value of n choose k. So let n and k be two non-negative integers such that k is not greater than n. The number of k combinations of a set of n elements equals to n choose k that equals to the factorial of n over the factorial of k times the factorial of n minus k. In our example five choose three or three combinations would be the factorial of five over the factorial of three times the factorial of two. Basically the factorial of five is one twenty over factorial of three is six times two that is one twenty over twelve and that is ten.

k-Combinations Problem 1.3

Now that we have the definition of what k combinations are let's try to write a program that reads from the user two positive integers n and k. Assuming that k is not greater than n. And it would print the value of n choose k. So an execution can look something like that. First we would ask the user to enter n and k. The user can say five and three for example and then the program would respond by saying five choose three is ten.

k-Combination Solution 1.4

So let's implement this program. So let's have our main. We are going to calculate n choose k by the formula of factorial n over k factorial times factorial of n minus k. So let's start first declaring n and k. Let's ask the user to enter n and k and read these variables n and k. Now we will need to calculate the factorial of n the factorial of k factorial of n minus k and do the calculation there. So in order to calculate the factorial of n we will define an accumulating variable n fact that would initially be one. And then we'll just multiply more and more values to it right. So we want to do one times two times three times four so on up to n. So we will use a for loop that would iterate one through n and each time it would multiply our accumulator this n fact by i. So first it would be n fact times one then n fact times two times three times four up to n. So eventually it would accumulate the entire multiplication of one through n. So at the end of this execution n fact would hold the factorial of n. Same thing we would do for k fact to calculate the factorial of k. So again k fact is initialized to one and then we would iterate one through k this time multiplying each value there. And same thing we will do for n minus k. So I named a variable n underscore k fact just because I can't use a dash in the variable name. But same thing n underscore k

fact is initialized to one and then we are multiplying one through n minus k all the values in that range. After doing that we are collecting we are calculating the n fact over k fact times n minus k fact and we name that value as k combinations. Now a point here is that I made k combinations be an integer but then I am dividing here some values maybe it could be a double. So if we think about it a bit more deeply we will see that it cannot be a double since we are using the division of two integers it would be counted as div. But then it would not be a double since mathematically we can prove that this value here represents n choose k or in other words the number of ways to choose k elements out of a set of n. And that is always a whole value that is not can't be 3.5 ways to choose k elements out of n. That is an integer value so we can feel safe to declare k comb as an integer and do the math here using div in this case. So after calculating k comb we can print that n choose k is k comb. So we've got that all figured out now take a look at these three parts here when we calculated the factorial of n the factorial of k and the factorial of n minus k. They look very similar to one another in one way and they are doing the same kind of calculation the same kind of process. Basically initializing an accumulator and iterating over a range each time multiplying the accumulator by the current value. Not only that it is basically calculating the factorial so you know let's that would be cool. Let's include the math library let's do something like include c math. And then instead of having n fact equals one and all of the for loop let's call the factorial function of math. Let's do n fact equals factorial of n. And same thing for k. K fact equals factorial of k. And same thing for n minus k. N minus k equals factorial of n minus k. And that would make our code first much more clear it basically speaks what we want to say. It says n fact equals the factorial of n we don't need to analyze a for loop and an accumulator to understand what they do. Basically we just read the words there. And yes so it makes it a lot clearer and we don't need to get into the details of implementation here. Unfortunately the c math library doesn't have a factorial function that's kind of unfortunate here. Luckily for us we are going to define our own factorial function just now. So yeah let's see how we do that.

Flow of a Program That Calls Functions 1.5

We want to create or define the function of fork Tauriel so we can use it three times here. No no we've already used functions we've been callers of function quite a few times already. We've called the square root function. You've called Also some other functions as well but that's going to be our first time that we're going to define our own function in this case it's going to be the fork Tauriel function.
So we want to define a function named fork Tauriel this function should get an integer as an input a number as an input should do some stuff and eventually return the fork Tauriel of number which is also an integer value in order to define this thing here a function that gets an integer and returns some integer we first defined the header of the function in a simple slot it looks like that in fact Oriya and then and closing parenthesis in parenthesis. We have in numb. Basically the name of the function is factorial the second word here in this line that's the place where we have the name of the function in the prentices we say what are the variables of this function in this case a number which is of type integer. So we are expecting to get an integer value locally will name it numb and on the left here we have the integer which is the type of the value returned by this function the calculation result of this function is an integer. So you have like if you read it in order. It starts this function is evaluated to be in into the return an int its name is for Korea and it would get a numb as an input and then you have a block of curly braces that basically init you have to say what. Is the implementation how do you take this input numb and calculate the fork Tauriel of numb. Out of it and here we have to write the code to calculate numb for Victoria given a value to numb. So the code is very similar to what we had previously Let's have two local variables factor is and I initially the factor as the cumulate are here would be set to one and then we'll iterate over the range from one to Naam right we want to calculate number for Tauriel So we'll have to rate from one to numb. Each time Multiplying factor is by the current value that we are iterating over. So first will multiply by one and then by two than by three than before and so on up to

number. So after having these few lines executed factories would hold the factorial of numb. We want to return that as a result for that we use the keyword return that basically announces that as the output of the function so return fact or is that is the value the integer there is returned from this function so that is the definition of the factorial function. First we say that interface how to use this function call it for Tauriel pass an integer to it and then it should return an integer inside the curly braces we say what the implementation he is or basically how to calculate how to create this factorial resource for total integer out of the input number using this for loop eventually returning that value.

Calling a Function Code 1.6

If we have our main here in addition to the main since we are calling factorial in this case even three times. We will need to give the implementation of factorial the function factorial. So now we have two things in our program. The main which is by the way seems very much syntactically as a function we will see a few words about it in one of our future modules. So we have the main function and then we have the factorial function. And let's see how this program basically executes. So first we read n and k from the user cout please enter n and k cin n and k. And then we have a call to factorial basically jumping to the code of factorial assigning num the local variable num with the value of n doing the calculation of n factorial. And returning res back to the main back to n fact. After doing that we call factorial again with the value of k associated to num. Jumping to the factorial call doing the calculations and going back to assigning the resulted value to the factRes back to k fact. And then we are calling factorial for n minus k jumping to factorial calculating the for loop a third time now and returning the result back to n minus k fact. And then we can calculate the combination there of n fact over k fact times n minus k fact into k combinations and printing the output. So you can see that we are calling this function factorial several times. Makes it very tempting to define this factorial function. We implement the algorithm of calculating factorial only once but we can use it over and over. One of the advantages of using a function. Also the name factorial is another great advantage here because instead of having a for loop three times we are having word name factorial appear three times. Make it much more readable and clear exactly what we are doing here. And this way we call this function over and over and calculate the factorial of n of k and of n minus k.

(Supplemental)

Let's take a closer look here at this program. It seems like we are calling factorial over and over which is a great thing. Each time passing a different value as our local num. First time we are calling factorial for n second time factorial with k third time factorial with n minus k. So each time num our local variable and the factorial function gets a different value. For example if n and k are five and three first time we will call factorial with num equals five. Second time we will call factorial with num equals three and next in the third time we will call factorial with num equals five minus three which is two. And the factorial would be executed the right amount of iterations and each time when factorial is done it goes back to where we called it from and do the assignment. So basically each time we call factorial we jump to the beginning of the factorial function same position. And each time we return from factorial we go back to where we came from. So first time we called factorial we came back to the first assignment. Second time we called factorial we went back to the second assignment. And third time we called factorial we went back to the third assignment. You can see that using functions calling functions is some other way of controlling the flow of our program. It is not sequential right so when we are calling factorial we are not going right back or right to the next instructions. We are jumping to a totally different position in our code doing some instructions over there and then jumping back to where we came from. That happens or in order to do that we would need to figure out a way to manage this kind of control flow. So jumping to the beginning of the function that's easy. We always each call jumps to the exact same position in the

code to the beginning of the function. But in order to return to the right place we would need some kind of mechanism to save and to store the position where we would need to jump back to. In a few minutes we will see exactly how we do that.

Data Types, Expressions, Control Flow 1.7

So to put it in context you can see that functions are a new way of a control flow. So if like we have the sequential flow that our instructions are executed one after the other in a sequential order. Then we introduced branching where we could some executions do one lines of code and another execution jump to do a different set of instructions. Iterative statements that allow us to repeat the same instructions over and over. To jump back to a position we've already been in our code and execute the instructions over and over. Function calls are none of these control flows are a different order that we are executing our instructions. When we call a function we jump to a totally new position in our code. Do or execute the instructions over there. And when the function ends using a return statement we are jumping back to where we came from. So that is a totally new control flow and in order to get a bit deeper understanding of what is going on inside the computer there. Let me introduce you to an execution model named the run time stack. And that is basically what happens inside of the computer during the execution time.

Runtime Stack Execution Model 1.8

So let's take a look at this program here that we have same program that we had before. And let's execute or trace this execution using the runtime stack model. So assuming you have your memory here as we do a sequence of bytes a sequence of data. Let me show you how the runtime stack model works for executing this program. I believe you will get the hang of it as we go and you will be able to apply it to other executions as well. It would help you predict exactly what the output or how programs basically behave. So ok so we have our main program when we start executing the main we first have all our local variables stored in the frame of the main function. In this case we have n k kcomb nfact k fact n minus k fact. All of these variables we declared at the head over our main. So we have a place for all of these variables. You will see that the runtime stack helps us maintain the variables and manage the variables that are accessible in each in each position we are in the code. In the case of main these are the variables. Then for example let's start executing. So line four would print please enter n and k. Then we get to line five cin n and k let's assume the user enters three and five. So n would be equal to five. K would be equal to three. Then line six we have a call we have an assignment basically n fact equals the factorial of n. And in an assignment expression we first evaluate the right hand side and then the result would be stored in the left hand side of the assignment. So basically it is a compound expression that has two steps here. An evaluation on the right hand side and then an assignment to the left hand side. So when we are calling the factorial of n when we have a function call let me show you what happens in our runtime stack model. So there would be a few steps each time we call a function. First time is creating a frame for the factorial function. The frame would include the parameters the function has what is inside the parentheses in this case num. The local variables factRes and I. And one additional data which is the return address right. We said that the function when we call function we always go to the same place but when we return each time we need to return to a different position in our code. So we need to store we need to keep where we want to return when this function ends. So that's why in addition to the data the function uses it also stores where to return when the calculation is done. So first step when calling a function is creating the frame for the function in case of factorial we have num as our argument. Local variables factRes and i. And then the return address. Step two would be to evaluate the arguments in our calling environment. In this case n. So when we are in main n is five we are calling factorial with a value of five. So step two is evaluate the argument. Step three is associating

the argument with the parameters. So in this case associating num the factorial num with the argument value five. Also updating the return address to be six to the assignment here. After we created the frame evaluated the arguments associated the parameters with the argument values update the return address. Step four would be to jump to the beginning of the function to execute the code over there. So in this case we will set fact Res to one. By the way when we are inside the function block the only variables we can access are the function's frame variables. We can access the factorial variables we cannot access the main variables at this position. So the factorial body cannot access the main n k and so on. Ok so factRes is initialized to one and then we start iterating so initially I is one. Then we multiply factRes by one then I increases to two. We multiply factRes by two. I advances to three we multiply that by three. So factRes is six and we keep on doing that until factRes holds the one hundred and twenty or when we are done iterating over the entire range from one to num. In this case five. And when we come to the return statement return factRes. So that's a new statement. Let me show you how this statement is evaluated in our runtime stack model. So we first evaluate so again a few steps here. First we evaluate the return value the factRes in this case one hundred and twenty. Then we figure out where we need to return to. That is address six. And then we pop out the current frame and jump with a return value in this case one hundred and twenty back to line six. So we assign n fact with the return value with one hundred and twenty. So n fact would get one hundred and twenty.

Runtime Stack Execution Model 1.9

Kind of complex but let's do it in this case two more times. So line seven again an assignment expression. First we divided the right hand side. So that's evaluate factorial of K. That is a function call right. So we are expecting four steps here. First create a frame for this function. In this case it would contain the parameters num the local variables factRes and I. And return address right. Step two is evaluate the argument in this case K. K is three. Step three is associate the argument the parameter with argument value in this case num is three. Also associate the return address in this case we want to return to seven. Not six as before because this call should return to line number seven. And the last step step four is jump to the beginning of the code start executing. So factRes would be one and than we will iterate and iterate and iterate until we will have factRes with a value of six. And then when we come to the return statement we will need to return the value. Again a few steps there. First we will evaluate the value we want to return factRes which is six. Then we will figure out where we want to return to address seven. Then we will pop out the local frame the function that we've just ended. No need for that data anymore. And then we will jump back with the return value to the return address. We were jumping back with six that's our return value to line number seven. We have to complete the assignment so k fact would get the value of six. That ends the execution of line number seven. Line number eight I hope you get it after this third function call. So again an assignment we have a call to the function factorial. So we are creating a new frame for factorial containing the parameters local variables and a return address. We are evaluating step two evaluating the argument n minus k. So under the main's scope n is five k is three n minus k is two. Step three is associating the parameter with the argument value so num would be two. Return address would be updated also to be eight. And step four we are jumping to the beginning of the code. Same location right the beginning of the factorial function. We will do the execution there initializing updating factRes I and advancing all of that. And when we want to return the value here let's do the steps we need to take when we are returning a value. So return factRes we first evaluate factRes which is two. Then we figure out where we want to return to that is eight. Then we pop out the current frame and jump back with the return value two to line number eight. So we assign n minus k fact with the value of two. Then line number nine we would assign k comb with one twenty over six times two. Basically which is ten. Then we print that five that is the value of n choose three that is the value of k here in our local scope. Right we are in the main so we are using the variables of the main

frame. Is and then we print k comb which is in this case ten. That basically when we are returning zero we are basically just as all other return popping out the current frame and jumping back to where we came from. Back to the operating system. We will talk about operating system in one of our future modules we'll get into that in much more detail.

Program Implementation 1.10

Let's try to write this program on our computer and make it executable we need to do one more tiny thing there. So let's go to the computer and make it work.

So let's take a look at our code here. So we have the main function just as we had it previously. We also have the factorial function we implemented. I just added include IO stream and using namespace STD so we can use the IO capabilities cout and cin and all of these guys there. Let's try to execute it. Ok so I press the play button. Oh my God build failed. Ok something is not fine here let's take a look what the problem is. It seems like that problems are in line nine and ten and eleven. Basically the lines where we are calling the factorial function or our factorial function something is not properly ok there. But let's see it says use of undeclared identifier factorial. Sounds like the compiler for some reason doesn't recognize the factorial function and says that we are using something that is undeclared or undefined. But then it is defined right. Our factorial is defined down here. The problem is that when the compiler builds the executable it translates it line after line starting line one two three four and so on. So when we compiler reaches for example line number nine it doesn't recognize the factorial right. Because factorial was not defined before line number nine. It was defined only in line number nineteen. So at that point the factorial the compiler says wait a second I don't know what factorial is how can I translate it and make an executable a machine language code for calling the factorial. I don't know what factorial is. So there are two ways to solve this issue. First one is defining factorial before using it. Basically let's take the definition of factorial up here before the main right. Something like that. First you see that there are no these errors are gone. If we execute the program ok build succeed which is good. I can enter five and three and then it would say that five choose three is ten. As we intended it to be. That is not a recommended way to solve this issue because when we are going to have programs with many more functions it will be difficult to figure out what is the order we need to define the functions so they would be defined before we call them. When we have tons of functions it becomes a very complicated task to do. Sometimes it is also impossible when we have circular calls. Function one calls function two function two calls function three and function three calls function one back again. So in this case we don't have a right order to play the function so each definition would become before the function call. So the preferred way to solve this issue is not to mind about the order of the way we are defining the function. So in this case I can define the main before factorial this case call the function before I define it. But in order for assuring the compiler that the definition of the factorial function is head to come we are adding function declaration up here before the main. The function declaration basically contains the header the prototype of the function. Basically the header line of the function. The interface for using this function. Basically saying that the function is named factorial it is expected to get one single argument of type integer and return an integer as a return value. When we are copying this header line this prototype the syntax is to end it with a semicolon. In this case it would look something like that. So now when the compiler translates this code after including or extending with the definitions of IO stream and using namespace STD. We are telling the compiler you can be sure that we are going to define a function named factorial later on. And just for your information this function would be named factorial it would get an integer num as an input and it would return an integer as an output. Then when the compiler reaches let's say line number eleven and it has to compile and translate this function call factorial of n. It knows that it would eventually find a definition for the function factorial. So it knows

that there would be a function with the same matching name factorial and passing n an integer as a single argument that's also fine because this function is expecting to get one integer as an input. And assigning the value factorial n into a variable of type int that is also fine because we said that this function is returning an int. So basically this prototype tells all the compiler needs to know for using for calling for interacting with this function. So by having or by supplying this prototype up here the compiler can know that the syntax of this function call is valid. So it is ok and the compiler just keeps on translating the entire main function. And then line twenty one this promise we made is coming to live. So we are basically implementing the factorial function. So that is fine and we can execute this code. Now we can enter five and three and get the value back. So now that we know that we can call functions we can define functions and we can declare functions let's continue doing that. We said that there are a few advantages of using functions. For example defining the factorial function first made our code more clearer more readable right. We have the word factorial n instead of some for loop here. We can reuse this function over and over. In this case we used it three times. By the way another point we can make here see that we returned factRes. We didn't print we didn't cout factRes. It is a big difference. If in some way instead of return we cout it factRes that would be not a good behavior because it would print these three factorial values. We don't want that we want to assign it to variables right. Also if we print it we can then not assign it to a variable. So make sure that you see that we intended to return the value not to print it. Later on if you wanted to print it we could have printed it back in the main. You'll see that a lot of times we would return values from functions. Typically that's the way we use functions where we return the value. It is very rare that you see that functions do the interactions with the user and print some values to the user. It happens sometimes but most of the times we just return the values back to the caller and the caller would decide whether it wants to assign it to a variable or it wants to print it to the user. What about defining another function here? You see that this calculating here that calculates the k combinations here we are doing it only once. But we can think of scenarios where we want to repeat this calculation over and over. So maybe we should take this few lines of code here. These calculating of factorial of n factorial of k of n minus k and doing the math to get the k combinations and create a new function for that purpose. So let's create a function named k combinations. There are two inputs n and k so we should do int n and int k. Make sure that when you have more than one argument or more than one input for a function you separate the input with a comma. Just like when you declare more than one variable you separate it with a comma. With one exception here for functions if you have more than one of the same type you still need to write explicitly the type of each variable. So you cannot do something like int n comma k. You will need to say int n comma int k. So this function k combinations gets two inputs n and k. And given n and k it would return an int right. So we have a curly brace lock for the code the body of this function. And let's define it. Basically it would be this code here right. So let's take that from here put it here. Right we have some problems it says it doesn't know the nFact variable so let's take n fact k fact n minus k fact make them local over here. So the k combinations function given n and k basically declare three additional local variables. N fact k fact n minus k fact that are set to be the factorial of n factorial of k factorial of n minus k. And after having these values we should return this thing here. N fact over k fact times n minus k fact. That is the return value that we are returning. And then up here in our main we will see k combinations after reading n and k to be whatever k combinations would return when called with n and k. Something like that ok. Let's take a look here so we still have one issue here that we are using k combinations that is not defined. Yeah that's basically we are defining it after using it. Let's make a declaration let's the prototype of k combinations up here. That should solve this issue. Ok let's look at our code now. So first we have the declarations of factorial and k combinations. Just so we can use it before we define it. Then the main has actually only three local variables. N and k that we are reading from the user. And k combinations k comb actually here. That would hold whatever k combinations results to. And then we print n choose k is whatever k comb is. Right and k comb would be set to the calculation of k combinations hold with n and k. So again take a

look at this main it seems very readable now. So we are reading n and k we are calculating k combinations for these n and k and then we are printing the result. Right we don't need to do a lot in order to understand exactly what happens here. And then k combinations is implemented very straight forward right. Given n and k we are defining these three local variables that are set to the three values of factorial of n factorial of k and factorial of n minus k. And then we are returning this combination here. Then we also obviously define the factorial function as well. Even though we are calling the k combinations only one time it is still important to define these functions. First for increasing the readability of our program and then for reuse maybe in other programs as well. Let's just execute it and see that it works as expected. So build succeeded let's enter five and three and then we get that five choose three is ten. So it seems to be working just fine.

Scope of Variables Example 1.11

Ok let's take a look at another example and see how our local scopes basically work. So for example let's have a main program that has a local variable n. It prints the value of N and then it calls a function named func. Passing n as an argument. And then it prints the value of n after this function call. And func let's define a function here named func that gets an integer n as an argument. By the way it returns void when we have void as a return value it means that there won't be any value returned. Right you can see that when we call func we didn't say some variable equals func right. Because this function doesn't return a value only does some calculations but not returning any return value. You can see that there is no return statement at the end. So a function can return whatever type it wants it can return an int a float a double just any type we want. Or it could be a void basically saying there is no value to return. Ok so we have this function this function sets n to be four and then prints the value of n inside. Let's try to trace this execution. So before the function N is three right n is set to three so just after that it would print before func three. Then we call func with three basically jumping to the function position setting n to be four. So inside the function n would be four. And just after the cout we jump back to where we came from and printing the value of n after this function call. So it would print after func but I am not sure whether the value that would be printed is the updated value of n basically four or the value of n was before which is three. So let's in order to solve this issue let's use the runtime stack model and trace this execution in that model. That would explain the behavior entirely. Ok so we start with frame for the main containing the local variable n. Initially n at line two is set to three so n gets the value of three. Then we line three cout before func the value of n is three. Then we call the function func. So a function call we have a few steps we need to take. So first we create a frame for this function containing the parameter n. Local variables there aren't any and the return address. By the way it is not a coincidence but the parameter name for the function func and the local variable up the main are both named n. And you can see there are two separate instances of variables both named n. One in the main frame and the other in the func frame. So there are two different variables I believe you probably already can guess what would be the value printed after func is evaluated. Ok so step one was creating the frame so we have n in the return address. Step two is evaluating the argument so up in the main the value of n in the argument is three. Step three would be assigning the parameter with the argument value basically assigning the local func's n to be three. Returning address would be five right after this call. And then we are jumping to the beginning of the function. Now we set n to be four. Which n would be set to four? Obviously the local n that we are currently in the variable's scope. So our currently scope is the func frame so we are setting func's n to be four. And then obviously when we print inside func the value of n is four. When this function ends there is no return value but we do figure out that we want to return to

address to line number five. We pop out the current frame and at line number five we are back to the main's frame. So the original n didn't change its value it was three then locally for the func the local n changed from three to four. So when we print after func the value of n it would print three. Ok so then we return zero pop out this frame and end this execution. So yeah so each function has its own local frame that when we are executing it the variables we access are the local variable. We have access only to these variables we don't have access to frames that are out of variables that are out of our local frame. And we can have variables with the same name but there would be two separate instances because each of them is in a different scope. You can see that the runtime stack has dual role here. It both manages the frames the local scopes of each function. What are the variables that are accessible in that function and it also helps us to manage the flow of the program. To remember where to go back when the function ends.

Pass By Value 2.1

Ok let's take a look at another example. Let's define a main that would have two local variables A and B. A would be initialized to three B would be initialized to four. Then we print the values of A and B basically before function calls. So we say before A is probably three B is probably four. Then we call the swap function with A and B and then we print the values of A and B after this function call. The swap function just as its name basically implies it would swap the values. So given A and B as inputs it would swap these values. So let's see what would be printed here let's trace the execution. So initially the frame for main would include A and B the local variables that at line number two would be assigned to three and four. Then the cout at line number three would print that before A is three and B is four. And then we have the call for the swap function. Again note that the swap function doesn't return any value it is a void function that is given two integers. Let's follow the steps for a function call for the swap function. Step one is create the frame for swap basically containing the parameters A and B. Same names of variables for the main A and B and parameters of swap. They can be the same they can be different anyway they are separate instances. Local variable temp here that's also included in the swap frame. And return address so the step number one is create this frame. Step number two is evaluating the argument up there in the main. A and B are three and four that are in step number three associated to the parameter the local parameters A and B. So the local A and B are also containing the same three and four we had up in the main. Return address would be to line number five right we are currently in line number four so we'll return back to line number five. And then we just need to jump to the beginning of the swap function. So in order to swap the values we would need to use this temporary variable to store the original value of A which is three so we can override variable A with the new value the swapped value of A. Then we say A equals B basically erasing the value the previous value of A putting four in its place. And then luckily for us we stored the previous value of A in temp so we can set B to be the previous value of A basically temp. Right so as you can see A and B did swap their values instead of three and four they are now four and three. But unfortunately the swap of values was locally in the swap function so when this function ends and we pop out this frame and we jump back to where the main is. To the main frame. The swapped values are already gone and A and B the original A and B of the main have their original values unchanged which are three and four. So this thing here would print three and four. And then the main ends and so on. Not surprising basically very similar to what happened in our previous example. But a bit concerning because we do want to be able to do some thing in a function and affect other locations as well.

Parameter Passing 2.2

Let me show you how we can deal with that. I want to introduce you a few ways a few types to pass parameters to function. There are basically two major ways for a parameter passing to functions. One of them is called call by value the second is call by reference. We've used up to now without mentioning it that's the only way I showed you how to create parameters and pass arguments to parameters we used call by value. Now I want to show you what call by reference is and what's the difference between both of them. So we'll talk about these two types in two dimensions. First I will talk about the syntactic difference what you need to do coding wise in order to call a parameter or to pass a parameter by value. And what you need to do syntactic wise in order to pass a parameter by reference. And then I will obviously explain what is the difference. What does it mean when you pass a parameter by value or by reference. So syntactically the difference is very simple. In a call by value you do exactly what we've done before. You have your func and then the parameter is just the type and the name of the parameter. Int x in this case. The syntax for call by reference is very similar but with one significant difference. So when you have func then you have int and then ampersand symbol x. So adding the ampsersand symbol next to the int that would make your parameter here be called by reference parameter. So that's the way you control the method you are passing your parameter. Either you just have the type before the name that would be call by value. Or type ampersand and a name that would be call by reference. So we know how syntactically to call or to pass an argument by value. How syntactically to pass an argument by reference. But what does it mean when we pass an argument by value versus passing an argument by reference? So the semantics is something like that the meaning is something like that. When we are passing a parameter by value we are evaluating the value of the argument and that value is passed. Where when we are passing an argument by reference we are evaluating the reference the position the location of the argument and the link or a reference to that position. That is what passes.

Example 2.3

I think it would be much easier to understand if we'll take a look at the swap example. So as we initially wrote it as you can see A and B these two parameter of the swap function were passed by value right. We have int A int B. Let's see what happens if we pass A and B by refenrece. Note that I added the ampersand sign symbol just for both A and B so now we have void swap and int ampersand A int ampersand B. Basically saying both our parameters would be called by reference. And let's see what the execution would look like. I would need to update the runtime stack model to support or to explain also call by reference. So it looks something like that. Again we start with the main frame containing the local variables A and B. Line number two assigns A to three and B to four. Line number three prints before A is three and B is four just as we had before. Then we have the call for swap and see what is different here when we call by reference ok. So again there are going to be a few steps. First step same thing we create a frame containing the parameters in this case A and B. Local variables in this case temp. And return address. Step two is a bit different in step two basically we evaluated the arguments. But since now we have two types of passing the arguments or two ways to pass the argument it matters whether we are passing an argument by value or by reference. In case we pass an argument by value we evaluate in step two the value of the argument. In case we are passing an argument by reference we won't evaluate the value we would evaluate the position the reference of the variable. In this case A and B are called by reference right so we won't evaluate the values of A and B which are three and four. We'll evaluate references links positions to A and B. The places where A and B are. And when step three we assign the parameters with the arguments we will assign a link to the main's A for A and a link to the main's B for B. So we are passing the references to the variables A and B to main's A and B. So the parameters the local parameters A and B of the function swap contain references basically links that's why I colored

them in blue just to make it look like a link. We are passing a link a reference to the main's A and B. Which when we look at this image now it seems that the swap function by using the local data in the frame of swap that has A B the local A B and temp. Now A and B have access to the main's A and B and maybe do the swapping the changes over there. So it seems to be good right. So again step one was creating the frame. Step two was evaluating the arguments in this case since they were called by reference the references to the variables to A and B. Step three is assigning the local variable the local parameters to these references. And step four the return address would be five. But step four would be jumping to the body of the function. And then we have first instruction temp equals A. What do you think would happen? Would temp contain a reference to the main's A or temp would contain the value pointed by the reference to A? So the semantics of C++ says that when you use reference variable in this case A and B. When you look at the value of this variable you always follow the reference. So temp equals A the value of A would be three. So temp would get the value of three. When we say in line twelve A equals B. What do you think would happen? Would the local A obviously it would affect the local A the swap A the swap's A. Would the local A contain a reference a link to B or would the main's A followed by the reference would contain the value four followed by the reference of B? So once again the semantics of C++ when we use referenced variables we affect the variables referenced by these variables. So the main's A would get the value of main's B that would set the main's A to be four. Again by using the local references we affect variables that are out of our scope. Which is great that's exactly what we wanted to do. And eventually line number thirteen would change the referenced B basically the main's B to be the value of temp which is three. So these few lines of code updated swapped the main's A and B values to be four and three. Obviously the fact that both the main's variables names are A and B and the local swap function variables A and B are the same that is just a coincident. It doesn't have to be like that. When the swap function ends this frame is popped out but when we go back to line number five we print that after the function call the value of A now is four and the value of B is now three. So it is a very powerful tool now that we have that passing by reference that when doing that would basically give the function permission to access variables and data that originally it doesn't have access to. Right when the main ends that pops out and everything basically ends.

Analyze Digits 2.4

Let's try to write another program. Let's write a program that reads a positive integer num and prints both the numbers of digits in num how many digits. And what's the total sum of all of these digits. We've done that a few times but now let's try to do it using functions. For example an execution could look something like please enter a positive integer. So user says three seventy five and then the program would respond by saying three seventy five has three digits right. Three seven five three digits. And their sum is fifteen three plus seven plus five.

Analyze Digits Introduction 2.5

When we implement this program we'll probably want to define a function analyze digits right. And let's think of the interface the input and output this function should have. So as an input for analyze digit I would expect some number num that's the input of this function. And this function should given the num do a lot of calculation a lot of yeah calculation there. And tell us two things first what is. Or not first one is what is the number of digits and the second is what is the sum of these digits. So there is one input and two outputs. And that is a bit of a problem in C++ because if you recall the syntax of a function we have the function name we can have as a parameters as many parameters as we want. Just separate them with commas say what's the type of each of them and you can have as many inputs as many parameters as you want. So for the input we can as many as we want but before the function we have the type of the return value. Basically saying that we can have only one value returned from a function. So when we use the return statement or the return expression we can only return one value. Which can

then be assigned to a variable or whatever. But we can have only one in this case we want to. So in order to implement such behavior we can work around that by using call by reference. Since call by reference basically can update a value in some other scope in for example the caller's frame scope. We can maybe instead of returning a value can update the variable that we want to set this return value to. So basically we can have we can write we can implement this behavior in a few ways. One of them would be say analyze digits input would be an integer num and then since we want to return two stuff one we would return as a return value for example the amount would be the int for the return value. And the second would be returned using call by reference variable. So we would have int ampersand int by reference. And out sum that would be the returned sum. By the way I note that the num the first parameter is an integer and it is called by value. You didn't have the ampersand right. And then the second parameter is out sum and it is called by reference. In a few minutes we will implement it and you will see exactly how that works. But just naming convention when a parameter is used for a return value I like to name it out something in this case out sum. You can also say ok if we can return values using call by reference let's do it this way. So let's have analyze digits with an input of num and have both values returned this way. So we would have out sum and out count digits. Both passed by reference so int ampersand out sum and int ampersand count out count digits. This case our function would be void right it won't return anything as a return value. It would return both of them in the mechanism of passing by reference. Let's implement the analyze digits using the computer. I'll implement the first one just to show the difference of how to return both by a return value and using a call by reference. Both on the implementation of the analyze digits function and in the calling of the function as well. So let's go to the computer and write this code.

Analyze Digits Implementation 2.6

So let's implement this program. First let's read the input so let's say please enter a positive integer end l. And then let's read it so let's declare a num variable here and read into num. Now we need to do the calculation here analyze digits so we would need two values to be calculated. The count digs and the sum digs. The count of the digits and the sum of the digits. We said to have a function analyze digits that as an input it would get num passed by value right. We don't want num to update anything. And it should return these two values ones as a return value as an int and the other by reference variable outsum. Ok so let's have this function here in a few minutes we will make the code here. But when we call this function it would be something like that. Let's call this function analyze digits right. The input would be this num here that we've just read. So this num here would be passed by value to our parameter num over here. Again I named them the same name but it is a coincidence they can have totally different names. And then this function given num should return these two values. The first one is the return value right. So let's have count digits store whatever this function returns right. This function returns a value that we can then store in a variable. So that's how we get the return value in the return into our count digs variable. The second would be get as a parameter that we would call by reference. So if we pass this sum digs local variable by reference. You see as the second argument is passed by reference. If we pass it by reference then by updating the value of out sum here locally inside this function that would update this main's variable with the value. So and I am just writing some non sense here but by doing out sum equals. I am just making up eight. That would update sum digs up here in the main to contain the value eight right. Because it contains a reference to this variable. So this variable would get its value by the assignment right. The assignment would put the value into count digs. And a local assignment inside the analyze digits function would update this value with a return value. So these or both of these variables would eventually have the right values in them. By two different mechanisms for returning values. So that would be the call for analyze digits. After we do that we can just announce that cout num the value of num and then has and then we have to say how many

digits. So count digs digits and their sum is and then let's have the sum digs value and end l. That's it right. That's basically it. So when calling analyze digits we assign both count digs with a return value and sum digs with a call by reference. Then we can assume that both of these variables contain the whatever the function calculated. Let's declare the function right. We have an error here let's declare this function. Semi colon at the end right don't forget. And the only thing we have left to do is implement the analyze digits function. So for that that's a very simple implementation. We basically iterate over the digits of num counting each one accumulating the sum of them. So eventually we would have a total count and a total sum. We've done that a few times already. So let's create a local count and sum variables. We'll initialize count to zero and sum to zero and then using a while loop. While num is greater than zero each iteration would have our current digit. So current dig would be num mod ten right. Let's also create current digit variable. So we have current digit and then we count this digit so count plus plus. And we accumulate this digit sum plus equals the value of current digit. Then we need to remove this digit so let's update num to be num div ten. Basically removing the ones digit. After we've done this calculation then these two local variables count and sum basically contain the two values we want to return. One of them should be returned. Return count. And one of them should be updated using this reference here. So outsum would be evaluated or assigned with a value of sum. Basically by this reference we are assigning sum digts with a current value of sum. One very important thing actually a return statement I don't know if I mentioned it explicitly but a return statement basically pops out the current frame and jumps back to where we came from. So all the statements that appear after the return statement basically are not executed. So the order that we return the values in is very crucial. We cannot first return this thing here and only after that update the reference variable. It is very important first to update the reference variable and only after that do the return statement right. Because otherwise this assignment wouldn't be executed. So after having both count and sum with right values we update using our reference the main's sum digits to be the sum and we return the count and this assignment here would update the coutn digits with this value. Let's just execute it make sure we didn't make any silly mistakes here. So we succeeded. Please enter a positive integer. Three seventy five and then it says that three seventy five has three digits and their sum is fifteen. Seems to be working fine. Just make sure you see how we used call by reference in order to work around the fact that C allows return only one value out of a function. We basically used call by reference in order to access our calling scope and update a value using that reference.

Solving Quadratic Equations 3.1

Ok as a final example let's write a program that solves a quadratic equation. Let's write a program that basically reads three real numbers representing the coefficients of a quadratic equation. And then it prints the solutions of the equation if there are any or an appropriate message. So for example an execution could look something like please enter coefficients of the quadratic equation. The user says one negative five and six. And then the program would say the equation one x squared just uses this symbol to says power up. So one x squared plus negative five x plus six equals zero solutions two and three. In this case there are two solutions. So before we go to implement this program let's go back to I don't know seventh eighth grade where we've talked about quadratic equations and make sure that we got the math all figured out. All the observations how to solve quadratic equations. So we know that we have this formula in case our equation A x squared plus B x plus C equals zero and it is a real quadratic equation. So A is not zero. Then we have an equation for the square root of these of the solutions of these x sub one and two are negative B plus minus the square root of B squared minus four A C over two A. See why it is important that A is not zero so we can divide by two times A. So this formula here is fine only when it is a real quadratic equation. And basically there could be different types of solutions basically depending on the sign of the discriminant. So in case B squared minus four A C is positive there

could be two real solutions here. For example a one negative five and six x squared minus five x plus six equals zero. We have two solutions x one and x two. But then if the discriminant is zero then when we take the square root it would also be zero and the plus minus zero would be the same. So for that case we would have only one solution. For example when A is one B is two and C is three you can check that B squared minus four A C would be zero and then we would get only one solution. X equals negative one. And there are also cases where the discriminant is negative in this case there is no square root real square root. So there is no real solution so we say that in this case there is no real solution. For example when A is one B is zero and C is one you will see that B squared minus four A C is a negative. In this case there is no real solution there are complex solutions but no real solutions. So so far for quadratic equations we either have two real solutions one real solution or no real solutions at all. But when we are reading these A B and C we are not guaranteed that A would be not zero. There are cases where A could be zero. In these cases the solutions can also be a bit different. So for example if A is zero B is zero and C is five so we get a zero x squared plus zero x plus five equals zero basically five equals zero no value substitute for x to get like a truth statement here. So in this case there is no solution at all not only not a real solution and you do have a complex solution. There would be no solution at all right. Or if all A B and Cs are zeroes so you'll get zero equals zero basically all values you'll substitute for x would give you a truth statement. So for zero zero zero for example you would get that all reals are solutions. So let's use these observations so there are five types of different solutions either two real solutions one real solution no real solution at all no solutions at all or that all the reals are solutions. Now that we've refreshed that let's go ahead and implement this program.

Implementation 3.2

Ok so let's implement a program that solves the quadratic equation. Let's also focus in this implementation not only making this program work but also breaking it down to functions in a correct way so that would increase the readability over a program and the usability of the functionalities we implement. So we can reuse it in other cases as well. Let's also focus on the how to document our code in this sense as well. So let's start obviously we would include iostream cmath we probably need a square root there so let's include the cmath. Using namespace std and let's also define constants for the different types of solutions. So let's add no solutions one real solution two real solutions all reals or no real solution. We'll have a number associated to each of them but the name of the constant matters the most because we are going to use the names to kind of speak what we want to say. Then we are going to define a quadratic equation right. I am before we start the main we have all of these definitions the includes the constants the prototypes here. So for example for quadratic we have double A double B D sorry double A double B and double C as three inputs all passed by value. And then quadratic should let's think what quadratic should return. It should tell us what kind of solutions we have here. Either no solution one solution two solutions and so on that is one thing it should return. And it should also tell us what are the solutions in case we have two real solutions we want to know what are the solutions. So we are expecting somewhere between one value only saying what the types of solutions we have to three. In case there are also some solutions. So in case there is no solution at all we'll get only one value back. In case there are two solutions we'll get two real solutions and then two values that are also as the solutions. So we are expecting somewhere between one to three values as a return. I think it makes sense to return the type of solution as a return value and to allow two parameter that could be passed by reference to store and to return in them the two solutions in case we have two solutions. For that I made the return value of type int and two doubles that are passed by reference so the quadratic can update the main's variables to contain or to have the solutions. That seems like a good prototype a good interface with a quadratic function. Let's also document what we kind of said over here. So that kind of document that describes the interface of with a function basically what the caller of the function should

know when it uses when they use this functions should come right here where we define the prototype of the function. So we said we say first that the quadratic function solves the quadratic equation of the form A x squared plus B x plus C equals zero. We should say what the input of the function is what the output of the function is and if there are any assumptions we should also say that. So the caller would know what assumptions this function basically takes. What assumptions the caller should have or should assume for this function behavior. So for inputs obviously A B and C are the coefficients of the equation. For outputs we should say that there are three kind of outputs even though two of the outputs are inside kind of the input location input position inside the parentheses. They are still considered to be output values. So for output we will say that there is one the type of the solution and I also say what mechanism I am using to return this output that is return value. And out x one and out x two are solutions to the equations these are output parameters called by reference.

Implementation 2 3.3

Let's go ahead and implement the main. Now that we've declared a function. Again in terms of documentation I am just describing what this program is supposed to do what input and output what kind of interaction this program would have with the user. So I am saying this program solved the quadratic equation input from user would be three real numbers representing coefficients of a quadratic equation. And the output to the user would be the solution to the equation if there are any or an appropriate message. The implementation of main is quite straight forward we'll define A B and C as local variables. We'll ask the user to enter the coefficients of the quadratic equation read it into these A B and C. And then we should call the quadratic function and figure out what type of equation we have here what kind of solutions we have and this kind of thing here. In this case so yeah we should start with announcing that the equation A x squared plus B x plus C equals zero has and then in this case I think the control flow that would basically branch between or choose between the different types of solutions is best implemented using a switch statement. So we'll do switch and the value we are switching over is basically whatever results with the call of quadratic. Basically the return value of quadratic. We said if you recall the return value that quadratic returns is one of the constants up there. The type of the solution right. So when we are calling quadratic with A B C x one and x two which are by the way two local variables that we defined here to store the optional solutions. But when we are switching over the call of quadratic basically the value of this call would be the return value. And that is the type of solutions. So we'll have a case for each type of solution. So we'll have case two real solutions case one real solution case no real solution case no solutions at all and case no reals. And we will also have like a default clause saying that there is an error because we are expecting to get one of them. So I hope you are convinced now that a switch makes it very readable. So we are switching over the call of quadratic and the different cases are basically the different kinds of solutions. Again the use of constants here makes our code very readable. We can basically see switch quadratic and case two real solution one real solution the different cases. And let's see how we behave in each one of these cases. So in case of two real solutions we just print the solutions are x one space x two right we said that in case there are solutions we want them updated in the output parameters. In this case x one and x two would contain these values. One real solution let's print one solution and then x one by the way how do we know that the solution is inside the x one? That's a thing we should maybe say a few words about it. No real solution by the way we just print no real solution or no solution at all or that all the real numbers are solutions.

Implementation 3 3.4

But let's add an assumption as we said to say that what's the behavior in case there aren't just two real solutions. So let's add these two assumptions. We'll write these assumptions right here in the

documentation so the user would know what kind of behavior to expect from the function. So the assumptions we are adding is if equation has one solution it would show in out x one. So when we in the main said that in case of one real solution we are printing x one we can be we can feel safe that x one would be the variable that would contain that single solution. Also we are saying that in case there if equation has no real solutions or no solutions at all the values of the output parameters are not defined. So we don't care what they have. And again in our main when there weren't no solutions we didn't access these output parameters. So by adding these assumptions to our documentation we basically tell our caller what type of behavior they should expect from our function to when they use it.

Implementation 4

I think we are ok now to implement or we can start implementing the quadratic function. So as we said the quadratic function gets three inputs as parameters A B and C. It would return the type of solution at our return value and it would also update out x one and out x two as the output parameters. For implementation basically we want to use the formula we have for the solutions of a quadratic equation but that only works when A is a non zero value. When we can divide by two A. So let's first make sure or calculate the cases where A is non zero. And then in these cases we can use the formula of x sub one two equals this thing here. Let's define the discriminant delta to be b squared minus four A C and check whether it is positive or equals to zero or else basically means negative. In case it is positive we'll have x one be minus b plus the square root of delta divided by two A. And x two would be minus b minus the square root of delta divided by two A. These are x one and x two are basically our local instances of the solutions then we want to return these two values. So we will set out x one and out x two to these two values and we should also return our type of solution for that case which are two real solutions. Right again make sure you first return the values in the output parameters and then do the return statement. Now let's take care of the case where our delta equals zero. In this case we said we are assigning out x one so let's first have x one be minus b over two A. Right the plus minus would be plus minus zero that's why we have only one. So out x one that's very important by our assumption out x one would contain x one and we return one real solution. And in case A is not zero but our discriminant is negative in this case we just say no real solution. We don't update out x one and out x two at all and we said we are not defining what would be returned by these values in case there are no real solutions. Ok now the last thing we have to take care of is what happens if A is zero. In this case actually our quadratic equation A x squared plus B x plus C equals zero when A is zero basically it becomes linear equation. B x plus C equals zero we don't have a quadratic element anymore. So in this case we should solve it as a linear equation. We can have the code here but actually I think it would make sense if we would create a function that deals with linear equations separately. Then we can use this function later on maybe not in this program but in another programs as well. So it makes sense to have or to define a function that not only not only to define a function that solves a quadratic equation but also to define a function that solves a linear equation. So the quadratic can use the linear but the linear can be used in other scenarios as well.

Implementation 5 3.6

Let's add another function here. Linear just as we had quadratic. The input for the linear would be actually linear has two coefficients so we would have double A and double B right. And the linear function should return the type of solution right maybe all reals we said in case of a linear. There could be no solution at all or no real solution or both of the times actually there is only one solution for linear equation. A x plus B equals zero. So just move the B divide by A and get one real solution. So let's have the solution as an output parameter and the type of solution as a return value. Let's document this function so linear first we say what is it intended to do so it solves a linear equation of the type A x plus

B equals zero. The input are A and B which are the coefficients of the solution. And the output is the type of the solution as a return value and out x one that is the solution of the solution of an equation or a solution of an equation as an output parameter. Our assumption here is obviously if there aren't any solutions then out x one is not defined.

Implementation 6 3.7

Assuming we have this function our quadratic function would now use this function. So in the else clause in case A is equal to zero then we just call the linear passing what would be the inputs for linear? So yeah so the quadratic B and C coefficients become the A and B coefficients for the linear. So we'll pass B and C and then out x one would be our reference to where we write the solution in case there is a solution. Maybe I'd note here that we are passing like out x one is already a reference to a variable x one in our main. And now we are passing it to another function by reference so we won't get a reference to the reference that would give us a reference to the original value in the main. So when linear updates out x one it basically updates the original reference we've passed from the main. So yeah so we're passing out x one and basically we are just rolling out the return value of linear as the return value of quadratic. So if linear returns no solution or one real solution or all reals we'll just keep on returning it as a return value of quadratic. That's why we are saying return linear.

Implementation 7 3.8

One last thing we have to do here is just implement a linear function that is very easy. So in case we have an equation A x plus B equals zero. So basically if A is not zero we can safely say that the solution would be again move B to the other hand of the equation and divide by A so X would be negative B over A. And then we'll set out x to be x and return one real solution. In case both A and B are zero that means that all reals are solutions. In this case I've also updated out x to be zero just as an example of A solution. And otherwise basically meaning that it is not or A is not different than zero it is zero. But it is not the case that both A and B are zeroes. So A is zero and B is not zero in this case we have zero x plus b equals zero there aren't solutions there. It's like zero x plus five equals zero. So in this case we just return no solutions.